AFIT/GSS/LSY/92D-4

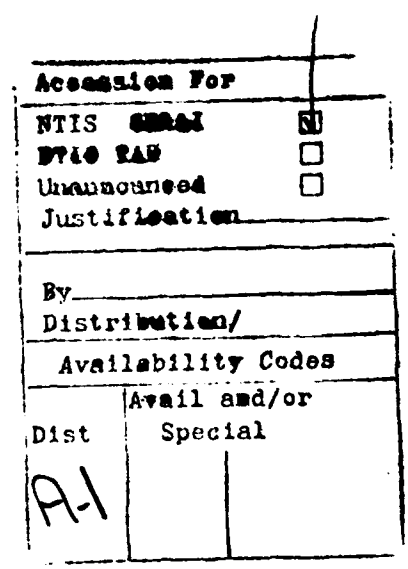
SOFTWARE SUPPORT MEASUREMENT AND ESTIMATING FOR ORACLE DATABASE APPLICATIONS USING MARK II FUNCTION POINTS

THESIS

Steven D. Radnov, Captain, USAF

AFIT/GSS/LSY/92D-4

The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.



AFIT/GSS/LSY/92D-4

SOFTWARE SUPPORT MEASUREMENT AND ESTIMATING FOR ORACLE DATABASE APPLICATIONS USING MARK II FUNCTION POINTS

THESIS

Presented to the Faculty of the School of Systems and Logistics

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Software Systems Management

Steven D. Radnov, BA Computer Science

Captain, USAF

December 1992

Preface

This thesis measures levels of Oracle database software support in terms of Mark II function points mapped to Oracle software components, and measures the effort and schedule response of four programming teams. The mapping allowed programmers to easily and unambiguously determine the size of the project before the coding began. A significant relationship was discovered between a given level of support and the programmers' effort response measured in work-hours as well as schedule response measured in calendar-weeks.

My education and insights gained from on-the-job experience in a database support organization inspired and sustained my obsession to see this through to fulfillment. My friends' and former colleagues' interest in my thesis has been a great source of encouragement. Captain Kathy Auzenne, Chief of the Systems Support Division sponsored this research and made it possible to collect enough real world data to put the theory to the test. Without observations from the field, most of the ideas would not have been tested. Thanks Kathy! Susan Zindorf's valuable insights into local software support activities and a 'reality-check' of the mapping of Oracle components to Mark II function points were pivotal. I wish she could have participated in the eventual data collection. Doug Burkholder and Beth Davis provided omniscient and omnipotent assistance as only DBAs can. Of course, I cannot forget Jeff Lindsey, Larry Frazier, and Kathleen Hale for their cooperation. Professor Dan Reynolds introduced me to a universe of possibilities in probability and statistics, as well as encouraged me in my endeavors. Thanks Dan! Finally, I would like to thank my advisor, Dr. Rich Murphy (a man of infinite patience).

The support and understanding of my family gave me the energy to persevere. My wife Carrie is a saint of course, for sacrificing, for listening, and for giving so much and asking so little. Our youngest daughter Alaina lights up every day with her endless smiles while her sister Natasha adds her boundless enthusiasm. They both look up to their big sister Rebecca, who charms every one she meets and makes my problems seem so small in comparison to her battle against Cystic Fibrosis. Rebecca is an inspiration to all . . . especially to me.

Steven D. Radnov

Table of Contents Page

Page

Page

Page

Page

List of Figures Page

List of Tables Page

This study investigated the results of measuring software support of Oracle database applications and estimating the effort and schedule required to provide support. Software measurement was accomplished with a variant of the function points metric, called Mark II function points, which is comprised of three weighted parameters, inputs, entities, and outputs. A technique for mapping Mark II function points to Oracle DBMS components was developed, and the size of the software support for each project, per team, was measured by tabulating and weighting the number of inputs, entities, and outputs that are added, changed, and/or deleted. Software support effort was measured in work-hours and schedule in calendar-weeks for given levels of function points. A data collection program was written to assist with tabulating the measurements and also provided an option for sizing the support by analogy. Observations were collected for 12 projects ranging up to 50 function points. The relationship between software support measurement in Mark II function points and the resulting effort or schedule was extensively analyzed for one and two person teams. A relationship determined by regression analysis was shown to be statistically significant for both effort and schedule.

SOFTWARE SUPPORT MEASUREMENT AND ESTIMATING FOR ORACLE DATABASE APPLICATIONS USING MARK II FUNCTION POINTS

I. Introduction

I.1 Background

A critical challenge for software support organizations is the management of changes to the supported system. Software projects are conducted according to a plan based on an estimate of the size of the project. The plan describes the expected schedule and effort needed to provide the software support.

The Air Force Institute of Technology has its own software support organization, the Systems Support Division (AFIT/SCV), that provides computer support to the Institute. SCV manages all of AFIT's software requirements for communications-computer systems including studies, analyses, requirements definition, design, development, documentation, testing, implementation, and training on unique software applications. SCV conducts software project status briefings for senior AFIT staff. They also assure that computer support contractors perform in compliance with all pertinent communications-computer systems regulations and the Quality Assurance Surveillance Plan (35:1). This research will focus on SCV's software support to applications, using a commercial relational database management system (DBMS) by Oracle corporation. The DBMS includes a database engine that relates records of data to one another and it provides several software tools to access the database. Some of the tools are SQL*Plus, SQL*Forms, and SQL*Reports.

The database query language that is used is SQL*Plus, which is an extension of the Structured Query Language (SQL) as defined by the International Business Machines corporation. SQL is a non-procedural data access language. The term 'non-procedural' means that the programmer does not have to specify the procedure for accessing data, but instead only has to specify *what* data is to be accessed (30:213). For example, there are no 'if-then-else' constructs and no means for making temporary computations.

SQL*Forms is a fourth generation application development language (4GL) that streamlines the software development process by generating most of the user interface input and output software details for the programmer, as well as data access. "Fourth generation languages are defined as those that are non-procedural in nature and end-user (requirements and specifications) oriented" (3:149). This correspondence

facilitates the partitioning of the software support requirement into meaningful software units of measure. A consequence of the ease of use of 4GLs is that they tend to restrict the variety of solutions, which creates a close correspondence between the source code produced by the applications generator and the software requirement itself. This allows the programmer to focus on assembling the functions that were requested by the user of the database applications. SQL*Forms produces interactive 'forms' that allow the user to store and retrieve data. SCV uses version 2.3 of SQL*Forms. Another tool, SQL*Reports, extracts records from the database and formats the output. SCV provides Oracle database software support by enhancing the forms or reports that access the database.

I.2 Specific Problem

Recent changes in the AFIT/SCV software support environment requires SCV to more effectively use available resources. Historically, SCV has relied upon expert judgement to estimate project schedules, but is now faced with tighter budget constraints, requiring the same level of support with fewer resources. The budget constraints have forced SCV to reduce its contracted support strength from seven to two analysts. To partially compensate, SCV had a net increase in enlisted programmer support from five to six, as a result of two NCOs leaving and three airmen arriving. The loss of the contractor support also represents a loss of the expertise on which project estimates are based, requiring newly trained enlisted programmers to make estimates. To meet this need, the Chief of the Systems Support Division has requested that thesis research be done in the area of software support measurement and estimating (34:151). Given a requirement to change an existing Oracle database application, SCV needs to be able to predict how long it will take to deliver the change, so that planning and contracting decisions can be expedited. Consequently, SCV would like to make accurate estimates of project size, effort and schedule.

I.3 Research Objective and Investigative Questions

The research objective is: to estimate the expected support effort and schedule given the size of an Oracle database software support request. The investigative questions that directed this research are:

- What is the most meaningful software size measure?
- How much effort and schedule is required for given levels of software support?
- What is the relationship between the size and effort or schedule?

I.4 Scope and Limitations

Only software support for Oracle database systems was tracked during the three-month data collection period of this thesis. During this period, SCV was staffed with new contractors and several new enlisted programmers, but no government civilian programmers. System accounting data relevant to project activities was not collected because automatic accounting is not activated in the SCV Oracle environment. The software baseline supported by SCV has no documentation or formal review process, and is managed as separate development and operational baselines.

I.5 Definitions for Measurement

- **Analogy**: characterizing the size of software to be developed relative to existing software.
- **Block**: SQL*Forms components that are based on database tables.
- **Calendar-week**: the period of time that a software support project was open from start to finish.
- **Entity**: "is anything (object, real, or abstract) in the real world about which the system provides information" (32:4).
- **Environmental factors**: any features about an organization or project other than size that may influence effort and schedule.
- **Field**: data items that are displayed on the screen or printed in a report.
- **4GL**: Fourth Generation Languages allow easy construction of screen-oriented applications.
- **Function Points**: a measure of the functionality of software that assigns points to software functions.
- **LOC**: an acronym for Line Of Code, which is a popular measurement of software.
- **Metric**: software measurement in general, such as LOCs or Function Points.
- **Non-procedural language**: a computer language for specifying what needs to be done, rather than how to do it (30:213).
- **SER**: Software Enhancement Request is a request for software support from AFIT faculty and administrative personnel to SCV.
- **SQL*Forms**: a 4GL that defines an interactive screen comprised of trigger steps, blocks, and fields.
- **SQL*Reports**: an Oracle database access language to retrieve and format data for output.
- **Work-hours**: the hours spent exclusively on the software support project, per programmer.

- **Tables**: data files that are organized into columns of entity attributes and rows of entity instances.
- **Trigger steps**: a SQL*Forms mechanism to validate inputs.
- **Virtual Machine**: a computing machine that could be implemented using hardware entirely, but instead simple hardware operations are used by software to provide complex operations.

I.6 Definitions for Estimating

- **ANOVA**: Analysis of variance.
- **COCOMO**: an acronym for COnstructive COst MOdel (6:58).
- **Deleted residuals**: a measurement of the i-th residual when the fitted regression is based on the cases excluding the i-th one (21:398).
- **Indicator variables**: quantify a qualitative variable for inclusion in a regression model.
- **Leverage**: a measure of the distance between the values of a given observation and the means of the values of the independent variable for all observations; it is used for outlier detection (21:395).
- **LSBF**: Least Squares Best Fit.
- **Randomness**: a lack of a systematic pattern.
- **Residual**: the difference between the observed value and the predicted value.
- **Specification**: the extent to which a regression model is appropriate for the data.
- **SSE**: Error sum of squares.
- **SSR**: Regression sum of squares.
- **SSTO**: Total sum of squares.
- **Studentized deleted residuals**: a deleted residual divided by its estimated standard deviation (21:399).
- **Type I error**: deciding to reject the null hypothesis when in fact it is valid.

I.7 Thesis Organization

Chapter One provides some background, defines the research objective, and defines some terminology relevant to this thesis.

Chapter Two reviews current literature on software measurement and estimating. The review focuses on two methods, the counting of lines of code and a very different method called function points.

Complexity measures and other metrics are also reviewed, as well as software cost estimating literature.

Chapter Three describes the software support measurement methodology. The three software support measurements that are collected are size, effort, and schedule. The sizing method chosen was Mark II function points mapped to Oracle database software components. The effort is measured in work-hours and the schedule in calendar-weeks. Software was written to collect the data and provide sizing by analogy, using data dictionary reports.

Chapter Four describes the software support effort and schedule estimating methodology. The prediction models developed are based on regression analysis of the programming teams' effort and schedule response to software support requests as sized by Mark II function points.

Chapter Five presents the regression analysis and the effort and schedule models for one and two person teams.

Chapter Six contains conclusions from the research and recommendations.

II. Literature Review

II.1 Overview

Prediction or forecasting of any kind is based on inputs. This literature review focuses primarily on the types of software metrics that are used by the software industry as inputs to prediction models. The primary software metrics reviewed and contrasted are lines of code and function points. Also, several complexity measures are discussed. Different environmental factors that may influence the software support were reviewed as well as existing effort and schedule estimating models.

II.2 Lines of Code (LOCs)

The most common measure of software size is lines of code. Rakos states that all software project estimating methods "are crucially dependent upon granularization: breaking things into small pieces" (25:128). Compared to a typical computer program, an individual line of code is a very small piece of the software. To understand an LOC, familiarity with the concept of a virtual machine is helpful. Tanenbaum describes different levels of virtual machines that exist on most computer systems. Each level has a code or language by which solutions to a problem are specified, and each level above level zero can be translated into lower levels (36:2–7).

The lowest level of virtual machines is the level zero, the digital logic level. At this lowest level, software is executed directly in the electronic circuits. At this low level, the semantics of the computer program are very difficult for most people to comprehend. At level one, the microprogramming level, a code is used to describe the computer's transitions through states of operation. This microprogram or microcode is represented by alphanumeric characters and special symbols. This level is much more comprehensible than level zero, but it is still cryptic and inconvenient to use. At level two, the conventional machine level, multiple microcode statements are represented by single machine code statements. This is easier to work with but still cumbersome. Level three is the operating system which groups machine code into operating system services. It is not until level four, the assembly language level, that substantial applications can be developed by a programmer. Most programming however is done at level five, the problem-oriented language level. At this level, English-like codes are used to specify the solution to the problem. These codes are displayed in [illegible], and the size of the program is measured in terms of the

number of lines of code, or LOCs (36:2–7).

The most quoted source on LOCs is Barry Boehm who uses the term 'source instruction'. This term includes all program instructions created by project personnel and processed into machine code by some combination of preprocessors, compilers, and assemblers. It excludes comment cards and unmodified utility software. It includes job control language, format statements, and data declarations. Instructions are defined as lines of code or card images. Thus, a line containing two or more source statements counts as one instruction; a five-line data declaration counts as five instructions (6:59). Boehm's is only one of many definitions of how to count lines of code after the code has been written.

Counting lines of code is popular because *after* the code has been written, there is a high correlation between LOCs and the effort that produced those lines of code. But a size estimate needs to be accomplished *before* the lines of code are written. Boehm takes it for granted that the software personnel can make accurate estimates of how many LOCs will be written to solve the problem at hand, but warns that underestimates can happen for three reasons. "First, people are basically optimistic and desire to please. Second, people tend to have incomplete recall of previous experience. Third, people are generally not familiar with the entire software job" (6:320–321). In addition to LOCs, the software support organization and software baseline are characterized by environmental factors.

Barry Boehm created a cost estimating tool based on LOCs called the Constructive Cost Model (COCOMO). In COCOMO, environmental factors are a way to differentiate the unique aspects of different projects and environments. The factors excluded from COCOMO are: type of application, language level, size measures other than lines of code, requirements volatility, personnel continuity, management quality, customer interface quality, amount of documentation, hardware configuration, security and privacy restrictions (6:345–346,475). The factors included in COCOMO are in four groups:

1. product attributes such as required software reliability, database size, and product complexity;
2. computer attributes such as execution time constraints, main storage constraints, virtual machine volatility, and computer turnaround time;
3. personnel attributes such as analyst capability, applications experience, programmer capability, virtual machine experience, and programming language experience;
4. project attributes such as modern programming practices, use of software tools, and required development schedule. (6:345–346,475)

The analyst provides inputs to COCOMO by characterizing the environmental factors on various scales and by estimating how many LOCs will eventually be produced. Then COCOMO estimates the effort and schedule of the project based on the analyst's inputs. However, the use of LOCs as a unit of software measure is the subject of much criticism.

Capers Jones points out the critical importance that measurement has contributed to the progress of science in general and then says that software measurement is probably the most deficient aspect of the field of software engineering. He is not saying that in the last 35 years of computer history no one has tried to measure software, rather he is saying that the accepted measurements are not measuring what they purport to measure. He specifically criticizes the LOC measurement for creating paradoxical information and he lists three problems (18:49).

First, there has never been a national or international standard for a LOC that encompasses all procedural languages. Some line counts are defined by physical carriage returns, while others are defined logically by delimiters such as a semicolon. The line of code counts can contain one, some or all of the following: executable statements, data definitions, comments, and blank lines. There is little agreement on how many times reused code should be counted: once, each time it is reused, or never. There's even more problems when different computer languages are mixed to together, and how to count additions, changes, and deletions (18:49).

Second, software can be produced by such methods as program generators, spreadsheets, graphic icons, reusable modules of unknown size, and inheritance, wherein entities such as lines of code are totally irrelevant (18:49). This is especially true of the 4GL SQL*Forms, where software is developed by interacting with the applications generation screen. Source lines of code are produced by the applications generator, but it is not necessary to look at them to complete a project. So if the programmer chooses not to ever look at them, for practical purposes they do not exist.

Third, LOC metrics paradoxically move backwards as the level of the language gets higher, so that the most powerful and advanced languages appear to be less productive than the more primitive low-level languages (18:49). To understand the productivity paradox, recall the concept of virtual machines. When a higher level machine groups together many instructions of a lower level machine, then that higher level instruction does the work of many lower level instructions but only counts as one LOC. For example, if a programmer using a lower level languages creates 1000 LOCs to solve a problem and another programmer using a higher level language solves the same problem with only 100 LOCs, then by LOC count the

first programmer is considered to have solved a problem ten times larger than the other programmer. When a fixed number of functions is delivered with increasing numbers of LOCS, productivity drops when measured in terms of functions per LOC, as depicted in Figure 1:

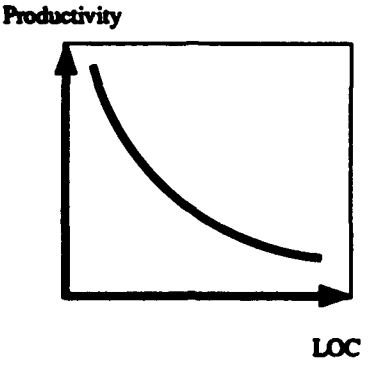


Figure 1. Productivity Paradox

Counting LOCs also leads to a quality paradox because high-quality programs are usually more succinct than programs that are hastily developed in an ad hoc manner. If the more succinct programs have fewer LOCs than the lower quality programs, an equal number of errors results in a higher error per LOC ratio than the lower quality program. In fact, if a lower quality program had five times as many LOCs than the more succinct program, then one error in the high quality program would have an error per LOC ratio equal to the lower quality program with five errors. In other words, if these two programs are offered to satisfy the same requirement, where one has five errors while the other has one error, then by using errors per LOC, they are deemed to be of equal quality. These paradoxes exist because the premise of using LOCs as a measure of software is that the LOCs *themselves* are the solution to a software requirement, rather than the functionality of the software.

II.3 Albrecht's Function Points

Function points are fundamentally different than lines of code. The premise of function point analysis is to measure the software *functionality* delivered to the user. The term 'functionality' shifts the emphasis away from *how* the LOCs implement the software, and instead emphasize what the software *does*. For example, if a payroll program is supposed to prompt the user for a social security number, locate that number in an employee record, and then display that employee's end-of-month pay, then the program

performs three functions: receives input, processes a record, and outputs the results. When counting functionality in this way, the LOCs used to implement the software are irrelevant, and the focus is on the software behavior. Continuing with the payroll example, if 10 programmers developed software to perform the same payroll functions, there would be 10 different LOC counts due to individual differences and the inherent variety associated with alternate software solutions. However, the number of functions provided by those 10 programmers would be identical. The software is the result of a request by the user to perform these functions, and function points are a measurement of the functionality required by, and valued by the user of the software. Allan Albrecht, an IBM researcher, introduced a means of quantifying software functionality by assigning points to five categories of functions based on their value to the user. Table 1 introduces the categories of function points.

Table 1

Function Count (FC)

Type	Description	Simple	Average	Complex
IT	External Input	x3	x4	x6
OT	External Output	x4	x5	x7
FT	Logical Internal File	x7	x10	x15
EI	External Interface File	x5	x7	x10
QT	External Inquiry	x3	x4	x6

- *Outputs* are items of business information processed by the computer for the end user.
- *Inquiries* are direct inquiries into a database or master file that look for specific data, use simple keys, require immediate response, and perform no update functions.
- *Inputs* are items of business data sent by the user to the computer for processing and to add, change, or delete something.
- *Files* are data stored for an application, as logically viewed by the user.
- *Interfaces* are data stored elsewhere by another application but used by the one under evaluation. (12:5)

Since Albrecht uses the functional value of the software as determined by the user as a guideline for subjectively identifying function points, there are significant ambiguities. To reduce ambiguities associated with assigning function points, Brian Dreger dedicated an entire book to enumerating rules for assigning function points. To identify outputs, Dreger says "count each unique data or control output procedurally

generated that leaves the application boundary" (12:10). To identify inputs, "count each unique user data or control input that enters the application boundary and also updates (adds to, changes, or deletes from) a logical internal file, data set, table, or independent data item" (12:16). Inputs and outputs are considered unique in so far as they have different formats or require different processing logic. Since the function point analysis is independent of any language, the measurement avoids the distracting and paradoxical details of lines of code (1:1). There are three steps in determining function points.

First, it is necessary to classify and count the five user function types delivered by the development project. These are external input, external outputs, internal file types, interface file types, and inquiry types. Each of the functions that are assigned to one of the five categories is further classified as: complex, average, or simple. The weights in Table 1 show how the total unadjusted function count (FC) is computed.

The second step is to calculate the processing complexity (PC) by ranking 14 environmental factors according to degree of influence on a scale from zero to five. The factors are: data communications, distributed data or processing, performance objectives, heavily-used configuration, transaction rate, on-line data entry, end user efficiency, on-line update, complex processing, reusability, conversion and installation ease, operational ease, multiple-site use, and facility change (2:640). The processing complexity adjustment (PCA) is:

$$PCA = 0.65 + (0.01 * PC) \tag{1}$$

Third, the delivered function points (FP) are calculated by multiplying the unadjusted function points times the adjustment factors (12:71):

$$FP = FC * PCA \tag{2}$$

Jones notes that after ten years of use, "the current national average for software productivity at the project level in the United States appears to be about five function points per staff-month" (18:11). Taking into account all the staff-months of an entire software development corporation and the function points delivered, Jones says a strategic or corporate productivity measurement averages 1.5 function points per staff-month. These numbers are for all types of projects and Jones notes that for management information systems (such as SCV's system), the productivity is higher than average at about eight function points per

staff-month. Jones also says that a maintenance programmer can take on support of a baseline ranging from 500 to 1500 function points, depending on the structuredness of the design (18:81,147).

Charles Behrens studied 25 development projects in the data processing organization of a large financial institution. He chose an hour as the unit of time and then calculated the output in function points per hour and also the unit cost in hours per function point. In 1980, 11 projects ranged in size from 27 to 599 function points and from 600 to 28,700 hours. Productivity measures ranged from 9.7 to 47.9 hours per function point. The mean was 18.3 hours per function point. In 1981, 14 projects ranged in size from 22 to 435 function points and 85 to 10,600 hours. Productivity measures ranged from 2.1 to 23.4 hours per function point with a mean of 9.4 (4:649). The interesting effect revealed in Behrens' research is that when unit costs are plotted against function points, the line of best fit is curving upward. The interpretation is that "as projects become larger, their unit costs become higher as shown in Figure 2, suggesting that everything else being equal, small projects are more productive than large projects" (4:649).

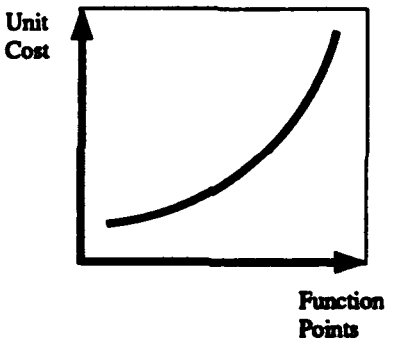


Figure 2. Behren's Research

Darlene Brown described methods of calculating productivity using function points and included maintenance activities. She chose a labor-month as the unit of time and calculated the ratio of function points delivered to the number of labor-months. She divided maintenance into two groups, enhancement activities and support activities. The difference between the two is that enhancement changes the functions while support does not change the functions but effort is spent to satisfy some user need. The measure for enhancement productivity is the change rate and for support is the support rate (8:27–28). Darlene Brown discusses the use of productivity rates as a means of predicting future work efforts. She distinguishes forecasts by new development, enhancement, and support. For each case she says that after the analyst has determined the number of function points that will be developed, enhanced, or supported, a simple

multiplication of the appropriate rate times the function points will give the number of labor months to do the job (8:30).

Several authors advocate that software metrics and estimation techniques should be tailored for a specific site. Ian Sommerville states that "the parameters associated with different models are highly organizational dependent" (31:516). Tom DeMarco offers the insight that, "software metrics cannot be treated as off-the-shelf products, and that organizations that adapt techniques and measures for their own use always seem to come out ahead" (10:160). Bryan Ratcliff and Anthony Rollo say that due to an organization's development paradigm function point analysis "will require substantial tailoring to fit an operational development framework" (26:80). Michiel van Genuchten conducted a survey of software engineers opinions as to why software projects are delayed and found that "the reasons were specific to the engineering environment in question because of differences among the software engineers, the type of software developed, and the organization of the department" (37:585).

June Verner and Graham Tate describe a case study for a development project that used a 4GL called ALL. The goal was to quickly build an information system for a correspondence school. They used function point analysis to estimate the size of the project, then they converted the function points to COBOL and adjusted for the presumed benefit of using a 4GL. They underestimated actual effort by 23 percent, but underestimated the schedule by only 3 percent (38:15–22). The results suggest that the analysts had a consistent measure of software size using function points before the code development.

Graham Low and Ross Jefery report that their studies indicate that "function points appear to be a more consistent a priori measure of system size than lines of code" (19:71). Low and Jefery report the results of an empirical research project into the consistency and limitations of function points as an a priori measure of system size compared to the traditional lines of code measure (19:64). They discuss some lines of code (LOC) attributes and variations and then state that using LOCs requires an "a priori estimate of system size based on past experience of the person performing the estimate with similar projects and/or systems" (19:64). The researchers go on to say that to the best of their knowledge "the consistency of LOCs as an a priori measure of system size has not been tested" (19:64).

The concept of counting function points has been studied extensively, but not without criticism. Boehm notes that function points suffer in terms of clarity and objectivity (6:482). Charles Symons analyzed the use of function points on some large projects and reported some problems with ambiguity. He devised a new approach called Mark II function points (32:5).

II.4 Symons' Mark II Function Points

Symons thinks that Albrecht's components are difficult to identify in a relational database environment, and difficult to interpret for on-line interactive transactions on the same screen. He questions how Albrecht arrived at the weights and levels of complexity when counting unadjusted function points. Symons does not think that the list of 14 environmental factors is exhaustive and that each of them should not carry the same degrees of influence. Also, he criticizes the lack of a measure of the internal processing complexity and the treatment of system components as discrete rather than integrated (33:18–22).

Symons has augmented the function point concept and calls it Mark II function points. It is based on a similar premise of measuring functionality as Albrecht's function points, except that Symons' method is more oriented towards development effort, than user value. Instead of five categories, Symons has three: inputs, entities, and outputs. The input and output categories also include Albrecht's interface category. For the new entity category, he applies entity-relationship analysis. For the purpose of counting function points, entities can be thought of as a combination of Albrecht's files and inquiries.

Identifying the complexity of each component is addressed by "counting the number of entity-types referenced by the transaction-type" (32:5). Referenced means created, updated, read, or deleted. The entity model in Figure 3 shows how entity types relate to each other and the lines with three prongs at one end indicate a one-to-many relationship between entities, (for example, a single customer can be associated with many orders).

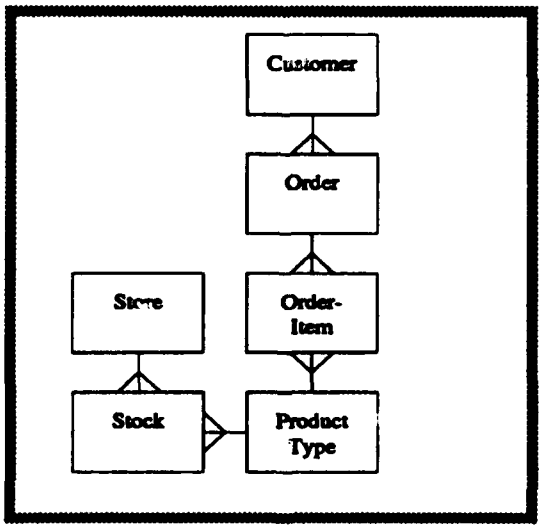


Figure 3. Entity Model (33:26)

Symons' 'transaction' construct, is "a unique input/process/output combination triggered by a unique event of interest to the user, or need to retrieve information" (33:23). Table 2 shows an example of function points counts by transaction. Based on the tabulations of inputs, outputs, and entities, the Mark II formula

Table 2

Unadjusted Function Point Count Example (33:26)

TRANSACTION	INPUTS	ENTITIES	OUTPUTS
Add Customer	53	1	3
Inquire Stock	2	3	10
Add Order Head	20	2	12
Add Order Item	6	5	6
Inquire Quantity	2	4	4
Report Stock	1	3	21
TOTAL	84	18	56

for information processing size in unadjusted function points is:

$$UFP = W_I \times N_I + W_E \times N_E + W_O \times N_O, \text{ where} \quad (3)$$

- N_I = the number of input data element types,
- W_I = weight of an input data element type,
- N_E = number of entity-type references,
- W_E = weight of an entity-type reference,
- N_O = number of output data element types,
- W_O = weight of an output data element type, and
- N_I, N_E, and N_O are each summed over all transaction-types (33:30).

Symons added five environmental factors to Albrecht's 14. The additional five factors are the needs: to interface with other applications, for special security features, to provide direct access for third parties, for documentation requirements, and for special user training facilities, such as a training subsystem. (32:7). These additional five factors added to Albrecht's 14 create a total of 19 factors. An additional 20th

factor emerged later, as the need to define, select, and install special hardware or software uniquely for the application (32:7). Symons calls these factors the degrees of influence (DI).

After analyzing 32 systems, Symons obtained weights which he calls the 'Industry Average Set': W_I=0.58, W_E=1.66, W_O=0.26 (33:30). The resulting unadjusted function points (UFP) formula is:

$$UFP = 0.58 \times N_I + 1.66 \times N_E + 0.26 \times N_O \quad (4)$$

The function point (FP) count is adjusted by the sum of the degrees of influence in the technical complexity adjustment (TCA) (33:80):

$$TCA = 0.65 + 0.005 \times \sum DI \quad (5)$$

$$FP = UFP \times TCA \quad (6)$$

Capers Jones has praise and criticism for the Mark II Function Points defined by Symons. Jones thinks that Symons' idea of counting entities and relationships adds a new dimension of rigor to function point counting. However, he also thinks that Symons' shift away from Albrecht's user value basis to a development effort basis is a step backwards. Jones also says that Mark II Function Points can result in counts that are up to 30% higher than Albrecht's function points (18:97).

II.5 Jones' Feature Points

A technique called feature points was created by Capers Jones. He supplements function points as defined by Albrecht with a sixth parameter for real-time and systems software, by categorizing the complexity of the algorithms (18:9). There are many definitions of algorithms, but for the purpose of counting, Jones states that "an algorithm is a bounded computational problem which is included within a specific computer program" (18:84). He also says the process of assigning the complexity of algorithms is ad hoc and is in need of a rigorous taxonomy.

The new algorithmic parameter is assigned a weight ranging from one to ten with a default weight of three. Also, the weight for logical data files from Albrecht's average value of ten is reduced to seven, reflecting the lesser significance of logical files to systems software. Including algorithmic considerations generates higher feature point counts for systems software than for MIS software. Symons praises Jones for introducing a measure of algorithmic complexity, but warns that it is difficult to establish standard categories

of algorithms: "Much work has been done to introduce measures for the complexity of algorithms, but the data to measure their 'size' are not generally available early in the system life in a form suitable for a function point metric" (33:42).

II.6 Halstead's Software Science

An assembly language programmer named Maurice Halstead developed a metric he called software science. He wanted to count: the number of errors, the effort to understand a program, and the effort to encode a program (3:1–2). He based his counts on operators and operands. First, size is computed:

- n_1 = unique operators,
- n_2 = unique operands,
- $n = n_1 + n_2$,
- N_1 = total operators,
- N_2 = total operands,
- size $N = N_1 + N_2$.

Halstead then used this size computation to develop a formula for the volume, V, which characterizes the encoding necessary for a particular program. He also defined potential volume, V*, as the minimum encoding of a program, but he offered no formula for V* (3:1–2).

$$V = N \times \log_2 n \tag{7}$$

He used V and V* to characterize the level of abstraction of a program, L, but since the value V* is not available, an approximation for L was devised:

$$\hat{L} = \frac{2}{n_1} \times \frac{n_2}{N_2} \tag{8}$$

Halstead also characterized the effort, E, to understand a program as the number of 'elementary mental discriminations' divided by the level of abstraction, implying that the more abstract the program, the easier it was to understand:

$$E = \frac{N \log_2 n}{L}, \text{ estimated by } \hat{E} = \frac{N \log_2 n}{\hat{L}} = \frac{V}{\hat{L}} \tag{9}$$

Based on an article by a psychologist that said humans process between five and 22 elementary mental discriminations per second, Halstead's experiments seemed to confirm that hypothesis with a value of 18.

Consequently, he used the value to compute the time, T, to encode a program:

$$T = \frac{\hat{E}}{S} = \frac{\hat{E}}{18} \tag{10}$$

II.7 McCabe's Cyclomatic Complexity

Thomas McCabe devised a measurement, v(G), for characterizing the difficulty of testing software and it is called 'cyclomatic complexity':

$$v(G) = e - n + 2p \tag{11}$$

The letter G is a graph representing the paths of flow in a program, e is the number of edges in the graph, n is the number of nodes, and p is the number of fully connected components (20:308). A rule of thumb is that a cyclomatic number greater than 10 characterizes a program that will be difficult to test. However, 'case' statements tend to drive up the cyclomatic number, and they are subjectively considered an abstraction that makes programs easier to understand (and therefore easier to test). Despite the apparent simplifying effect of case statements, they create many paths that require consideration for testing purposes, and so the high cyclomatic number does characterize that testing problem.

II.8 Other Sizing Methods

There are a variety of other measures of software. Henry and Lewis present "an experiment that introduces a non-destructive method for integrating metrics into a large-scale commercial software development environment" (16:89). They discuss three categories of metrics: code, structure, and hybrid metrics. Code metrics typically measure the source code instruction within a single module and have limited value for the total software system. "Structure metrics concentrate on the overall structure of a software system by evaluating the inter-connectivity among procedures" (16:90). Henry and Lewis used structure metrics called Kafura's information flow and Belady's cluster metric.

Henry and Kafura characterized the complexity of an entire system of interacting modules (15:510–518). They count the flows into and out of a module as fan-ins and fan-outs. The information flow (IF) metric is based on the fan-in and fan-out of the modules, and the length of the module which

is some complexity metric of the measurer's choice. The information flow (IF) is:

$$IF=length \times (fanin \times fanout) \quad (12)$$

Henry and Lewis also discussed hybrid metrics which are code and structure metrics combined. They call the heart of their structure metrics the communication database because it "identifies communication lines among modules" (16:90). This is similar to a data dictionary. Their communication database is "used to inform a system developer of the proper order in which to rebuild the components of a system, based on the use of specific modules by other modules" (16:91). This is critical information for estimating the scope and effort of a work order, because a change to one module may require a change to other modules that use the first one being changed.

Another aggregate metric is measured across time. Henk Blanken discusses "query processing in a database management system (DBMS) that has complete control over versioned complex objects" (5:1). These objects are called complex because in addition to the complex relationships between data structures, complex objects record a history of the changes to the object. From the point of view of someone who manages the software configuration of modules, such a database could store source code and the history of changes to it. Blanken discusses what he calls 'as-of-queries' and 'walk-through-time' queries (5:2). An as-of-query retrieves information that was current as of a certain time and date. The more relevant query for this literature review is the walk-through-time (WTT) query. He states that a WTT query could be constructed "by generating an as-of query for each moment of change" (5:2).

II.9 Estimating Effort and Schedule

Software cost estimating models have a reputation of being inaccurate. Boehm rates a software cost estimation model as good if it is within 20 percent of actual costs 70 percent of the time, within classes of projects that were used to calibrate the model (6:32). The predictions are based primarily on the size of the project, along with a variety of other factors. John Rakos states that "there are only two factors that affect the duration of a task: the complexity of the task and the productivity of the personnel performing it" (25:132). Most models like COCOMO allow a personnel factor to be characterized and submitted as input to the model. Considering the interpersonal dynamics involved in software development, a team of programmers can have a performance response different from another team, so it seems reasonable for an

organization that is tailoring its own model, to track the performance of specific teams as if they were individuals. Humphrey reports a method for projecting software productivity by collecting data for specific individuals or for a specific group of individuals working together as a team. He also points out that individuals' productivity follows the individual's learning curve, and so successive development times are autocorrelated. He cautions that the collected data is unique to an individual or team:

> It should be emphasized, however, that if the database concerns the productivity experience of one programmer, the projected productivity values are only valid for that individual. If team projections are desired, relevant team experience data should be used (17:196).

Lawrence Putnam defined methodologies for estimating effort and schedule for software projects with different numbers of programmers. He describes small, and medium to large projects, with medium projects having four or more people. Putnam says that there is enough similarity in the group problem-solving process that groups of four or more people tend to have similar problem solving responses regardless of team composition. However, for a small group of one, two, or three people, there is greater variation in problem solving response from group to group due to individual differences (24:216).

II.10 Summary

The quality of a prediction can be no better than the quality of the inputs to the prediction process. Existing software effort estimation models use size as the primary input. This review focused on software size measurements such as lines of code and function points. The essential difference between LOCs and function points is that LOCs measure the solution to a software requirement, while function points measure the functionality in the requirement itself regardless of how many LOCs were created. Other popular sizing methods focus more toward measuring the complexity of software based on some form of software unit interrelationships.

In addition to size, many environmental factors are quantified as inputs to effort estimation models. These factors are intended to characterize the uniqueness of a given software organization for software cost estimating tools intended for industry wide use. Finally, case studies were reviewed that explored the problem of establishing programmer productivity and predicting project duration.

III. Measurement Methodology

III.1 Overview

Data collection methods were developed to satisfy the research objective: to estimate the expected support effort and schedule given the size of an Oracle database software support request. This chapter addresses the investigative questions related to software support sizing, effort, and schedule measurement:

- What is the most meaningful software size measure?
- How much effort and schedule are required for given levels of support?

The guiding principles for selecting a meaningful sizing method is to collect data in a way that is:

1. unambiguous and
2. easily estimated before the project begins.

In Chapter Two, Rakos noted that software project estimating depends upon breaking software into small pieces, but if the pieces are too small the project staff could spend an inordinate amount of time trying to identify every last piece. The benefit of fine granularity should not come at a cost that is a large proportion of the project itself. Consequently, there is a trade-off between having less ambiguity in the sizing definition at the same time as easy estimation before the project begins.

III.2 Software Unit of Measure

Oracle SQL*Forms and SQL*Reports have little if any algorithmic complexity, and consequently all of the various complexity metrics are of little use. Since SQL*Forms and SQL*Reports are represented by source lines of code, a look at LOC counting was warranted. Upon closer examination LOCs are not very meaningful. First, they are very ambiguous as highlighted by Capers Jones, and Barry Boehm even says that, "source instructions are not a uniform commodity, nor are they the essence of the desired product" (6:32). In fact, they are even less meaningful in the case of SQL*Forms, because the LOCs are normally machine generated and possibly unseen by the programmer. The literature indicates that, before the project begins, it is easier to estimate function points than LOCs (19:1). The close association of function points with the functionality required of the software increased its appeal as an independent variable characterizing software size. However, the ambiguous separation of inquiries and interfaces from inputs and outputs as defined by Albrecht leaves a lot of room for erroneous categorization, and is therefore less objective. Capers

Jones points out that although Dreger's book successfully establishes consistent rules for counting function points, Dreger varies from the way Albrecht counts. "Notably, Dreger tends to accumulate slightly more kinds of things for inputs, outputs, and inquiries than would be normal when using the regular (Albrecht) IBM method" (18:98–99). Besides, having to master all the rules that fill a book is not a good candidate for making easy size estimates before the project begins. Also, Albrecht's category for logical files is not easily applied to relational databases. Symons list other ambiguities. He notices a problem with on-line, interactive transactions, where the same screen processes inputs and outputs, and he asks several questions:

- Is the screen to be counted as an input or output or both ?
- Are the logical file references (which we need to know for determining complexity) made from the input or the output, or both ?
- Is a retrieve-before-update the same as an inquiry ? (33:19)

Of all the sizing techniques reviewed, Mark II function points appear to be both unambiguous and easy to estimate before a project begins. For these reasons, Symons' distinct categorization of inputs, outputs, and entities will be used.

III.3 Oracle Software Measurement

The first step is to identify the Mark II inputs, entities, and outputs involved with a given Software Enhancement Request (SER). Inputs comprise that part of the software associated with display preparation and data entry validation. Outputs are associated with preparation for display or printing. Entities involve all work between the inputs and outputs, such as database accesses and data manipulations (33:87). Symons summarizes:

> The task then is to find properties of the input, process, and output components of each logical transaction type which are easily identifiable at the stage of external design of the systems, are intelligible to the user, and can be calibrated so that the weights for each of the components are based on practical experience (32:4).

For the Oracle database management system, SQL*Forms and SQL*Reports components were mapped to function points as defined by Symons. A trigger step is associated with data entry validation and cursor movement, so it was mapped to Mark II inputs. A SQL*Report did not have inputs counted since it is not used for data entry into the database. SQL*Forms are based on fields that define how the output will look, so fields were counted as Mark II outputs. SQL*Reports use 'select' statements to

extract database entries for subsequent output, so 'select' statements were mapped to Mark II outputs. SQL*Forms have one or more blocks that are based on database tables or entities, and are used to retrieve and manipulate data. The blocks were counted as Mark II entities. SQL*Reports are also based on tables, so tables count as Mark II entities. A single SQL*Form itself was counted as a Mark II transaction, as was a SQL*Report. This mapping Mark II function points to Oracle components shown in Table 3 satisfies both guiding principles for selecting a measurement technique stated in the overview:

1. unambiguous and
2. easily estimated before the project begins.

Table 3

Mapping Oracle Components to Mark II Function Points

Mark II Terminology	SQL*Forms Terminology	SQL*Reports Terminology
Transaction	A form	A report
Input	Trigger-step	n/a
Entity	Block	Table
Output	Field	Select

First, blocks, fields, and trigger-steps are sufficiently unambiguous that one cannot be confused with the other. Second, the mapping allows for easy estimates because the programmer can characterize software size in terms of familiar and *meaningful* Oracle components. The programmers provided the function points count after both the analysis and the design were completed. The environmental factors are considered stable and not varying across projects, and so were not used to adjust the function points. An illustration of the functional model for counting is shown in Table 4, which revises Table 2 on page 15 in Chapter Two by substituting SQL*Forms and SQL*Reports terminology, with the Mark II terminology:

Table 4

SQL*Forms and SQL*Reports Function Point Count

SQL*FORMS or SQL*REPORTS (transactions)	TRIGGER-STEPS (inputs)	BLOCKS or TABLES (entities)	FIELDS or SELECTS (outputs)
Add_Customer	53	1	3
Inquire_Stock	2	3	10
Add_Header	20	2	12
Add_Item	6	5	6
Inquire_Quantity	2	4	4
Report_Stock	1	3	21
TOTAL	84	18	56

III.4 Effort and Schedule Measurement

Symons measures effort and schedule in units of work-hours and calendar-weeks. He defines work-hours as "one hour of work by one person, including normal personal breaks, but excluding major breaks such as for lunch" (33:84). The programmers recorded the number of hours spent exclusively on a given project per person. Any hours spent on activities other than the project were not included in the project's work-hours. For a two-person team, the team's work-hours were the sum of each programmer's hours. The measurements were accurate to one tenth of an hour.

The programmers recorded the number of calendar-weeks from the day the project began to its completion. The count of calendar-weeks included work on the project and any other demands upon the programmers. The count of weeks was for the project and did not vary by team composition as did the measurement of work-hours. The measurements are accurate to one tenth of a five-day week, or about half of a day.

III.5 Data Collection Program

A data collection program is listed in Appendix H, and was written to assist with data collection in two ways. First, the program will simply prompt for the size of the software support characterized in

terms of Mark II function points that are added, changed, or deleted. Second, the program provides the option of 'sizing by analogy' in the case of an entirely new transaction being added. Sizing by analogy was accomplished by scanning Oracle DBMS data dictionary reports. An example of a data dictionary report is listed in Appendix F. The data collection program is written in the Ada language. The program could have been written using SQL*Reports, thereby directly accessing the data dictionary at the same time as providing automated assistance with sizing and estimating. However, the expressiveness and generality of the language was uncertain at the beginning of the development of the program. With Ada, the development was uneventful due to the availability of a debugged library of reusable, generic components known as the Booch Components (7:71).

For purposes of sizing by analogy, the SQL*Forms source code will be reduced to the parts essential for counting, using a DEC VAX DCL command file listed in Appendix E, that invokes a regular expression parser, 'fgrep'. The parser reads search patterns from the file fp2.re, also in Appendix E. If any lines in the data dictionary report contain the search patterns, then those lines are written to a different file for the function point counting portion of the data collection program to read. An example of reduced source code for a SQL*Form is in Appendix E, and shows trigger steps, blocks, and fields. Symons suggests computing "*Mark II unadjusted function points automatically from a functional model of a system stored* for example in a data dictionary" (32:10). The output from an Oracle SQL*Reports program in Appendix G for extracting a functional model from the data dictionary is the input to the regular expression parser that reduces the data dictionary report.

When an analyst receives a software engineering request, analysis is performed and the extent of changes necessary are identified and collected. The analyst first invokes the Digital Equipment Corporation VAX/VMS control language command file that reduces the size of the source code reports to be used for estimating by analogy. When sizing by analogy, the data entry program will scan the reduced SQL*Forms source listing for Mark II function points. The software support requested by the SER will be broken out into additions, changes and deletions of inputs, processes, outputs and entire forms and/or reports. The data collection program will provide a prompt for sizing, estimating or reporting actual results. After choosing the sizing option, the analyst enters a one to five digit SER number, and each member of the team working on this particular SER. A brief explanation is displayed, requesting forms or reports to be added, changed or deleted to support the SER.

- *Added* means creating a completely new form or report.

- *Changed* means that within an existing form (or report), blocks, fields, and/or steps (or tables and selects) will be added, changed or deleted.
- *Deleted* means completely deleting an existing form or report.

The analyst is now prompted for one or more of the software support actions for one or more forms. If the analyst chooses to enter forms that need to be added, an explanation appears telling that the form size will be estimated by analogy to an existing form with which analyst is familiar. If the analyst chooses to enter existing forms that need to be changed or deleted, then prompts appear to receive the names of those forms. The program then displays a message telling the analyst that it is scanning the forms to count Mark II function points, and then prompts for changes within the forms that were identified as requiring changes.

First the program asks how many trigger steps, blocks or fields will be added. Next, it asks how many trigger steps, blocks, or fields will be changed. Last, how many trigger steps, blocks, or fields will be deleted. The program then returns to the main menu and at this point the analyst selects the Estimate option. The screen displays the size of the current SER in terms of Mark II function points. At the main prompt, the analyst can enter a question mark and receive elaboration of the data entry process. After the SER is completed, the analyst reports work-hours spent on changes and testing, and calendar weeks elapsed during changes and testing.

III.6 Summary

To satisfy the research objective, the method of answering the investigative questions related to software support sizing, effort, and schedule measurement is described in this chapter. As a result of the literature review the chosen software size measure is Mark II function points. The reason for this choice is that Mark II function points are both unambiguous and easy to estimate before a project begins, and meaningfully correspond to the software support requirement. The environmental factors within SCV are considered stable and not varying across projects, and were not used to adjust the function points. Mark II function points are mapped to Oracle database software components in SQL*Forms and SQL*Reports. Software assisted with data collection and sizing by analogy, using the data dictionary reports produced by an Oracle SQL*Reports program used by the SCV programmers.

IV. Estimating Methodology

IV.1 Overview

Several models for predicting project effort and schedule were constructed to satisfy the research objective: to estimate the expected software support effort and schedule given the size of an Oracle database software support request. This chapter addresses the investigative question related to estimating: What is the relationship between the size and effort or schedule? In addition to the selection of Mark II function points as the software unit of measure, Symons' terminology for effort and schedule units were also used, namely, work-hours and calendar-weeks (33:140–142). Descriptive statistics were used to focus the regression analysis. The software used for analysis was the Statistical Analysis System (SAS). Within SAS a procedure known as PROC CHART provided histograms for the data (28:69–95) and PROC REG performed the regression computations (29:773–876).

Having chosen Mark II function points as the metric, it would have been convenient to use a commercial off-the-shelf cost-estimation tool based on Mark II function points. A company named Strategic Systems Technology provides just such a software cost-estimating tool called Before You Leap II, or BYL II. Symons cites it for its ease of use and flexibility (33:177). Unfortunately, BYL II was not available for this research.

IV.2 Population

The populations of interest depend on the composition of the programming teams. If one individual is working on the project, then the population of interest is the individual's response to the size of the software support requested as measured in Mark II function points. If two individuals are working on the team, then the population of interest is the response for that particular pair of programmers. For a project requiring two programmers, the need for interpersonal communication and coordination adds an additional activity to the effort, and therefore a different population of responses than for a single programmer.

IV.3 Descriptive Analysis

The descriptive analysis tabulates and summarizes the observed teams' responses to given levels of software support requests as measured by Mark II function points. The response is measured in work-hours expended within a schedule of calendar weeks. The statistical software procedure SAS PROC CHART created histograms of the data collected, which are listed in Appendix A. Also, scatter plots graphically display the relationship between function points and effort or schedule.

IV.4 Regression Analysis

A regression analysis was performed to determine the nature of the relationship between the predictor (Mark II function points) and the response (effort and schedule) by means of a least squares best fit (LSBF) regression line. Figure 4 illustrates a simple regression plot relating function points to effort.

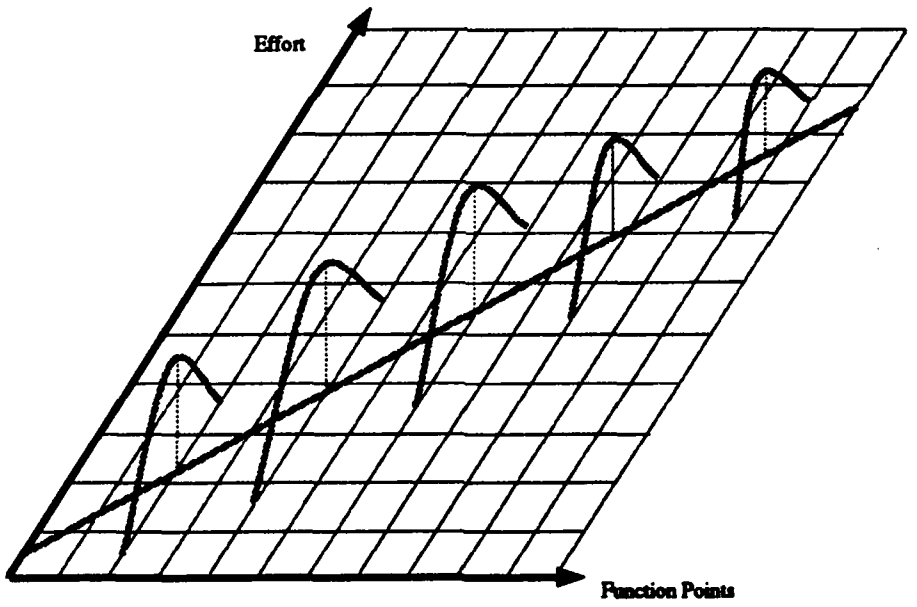


Figure 4. Effort vs. Function Points Plot

IV.5 Indicator Variables

To allow for different teams' responses to the same level of function points, indicator variables denote team composition in the regression models. Four teams (A, B, C, and D) would be denoted with only three indicator variables (I_1, I_2, and I_3):

Team	I_1	I_2	I_3
A	1	0	0
B	0	1	0
C	0	0	1
D	0	0	0

(13)

Team D is represented in this model when all three indicator variables are equal to zero. A model with three indicator variables would have to estimate a slope and intercept for each of the four teams, or eight parameters (21:351). With 12 observations though, estimating eight parameters reduces the degrees of freedom to four. Consequently, only one indicator variable was used to denote a two-member team versus a single programmer, preserving eight degrees of freedom for estimating four parameters. The indicator variable assumed the value of one for team D, and zero otherwise. Of the four teams, team D was subjectively determined to have responses likely to differ the most from other teams. Since team D is the only team with more than one programmer it has a line of communication between programmers that the other teams do not have. Teams A, B, and C represent three different individual programmers. These classifications lead to the initial models for effort and schedule.

IV.6 Effort Model

Work-Hours = $\beta_0 + \beta_1$ufp + β_2I + β_3Iufp + e.

- I=0 : Work-Hours = $\beta_0 + \beta_1$ufp +e, for teams A, B, and C.
- I=1 : Work-Hours = $(\beta_0 + \beta_2) + (\beta_1 + \beta_3)$ ufp + e, for team D.

IV.7 Schedule Model

Calendar-Weeks = $\beta_0+\beta_1$ufp + β_2I+ β_3Iufp + e.

- I=0 : Calendar-Weeks = $\beta_0 + \beta_1$ufp + e, for teams A and B
- I=1 : Calendar-Weeks = $(\beta_0+ \beta_2) + (\beta_1+ \beta_3)$ ufp + e, for team D.

Several statistics were calculated from the data and compared to the criteria in the following sections.

IV.8 F-Ratio

The risk of a Type I error was controlled at a significance level of 0.10 for the F tests.

- Ho: $\beta_1 = \beta_2 = \beta_3 = 0$
- Ha: not all parameters are zero.

Critical Value: The critical value for the F statistic is $F(\alpha=0.10,p-1,n-p)$.

Decision Rule:

- If $F^* \leq F_{crit}$ then do not reject the null hypothesis.
- If $F^* > F_{crit}$ then reject the null.

IV.9 Parameters

The risk of a Type I error was controlled at a significance level of 0.30 for the t tests.

- Ho: $\beta_1 = 0$, and Ha: $\beta_1 <> 0$
- Ho: $\beta_2 = 0$, and Ha: $\beta_2 <> 0$
- Ho: $\beta_3 = 0$, and Ha: $\beta_3 <> 0$

Critical Value:

The critical value for the t statistic is $t(\alpha=0.30,n-p)$.

Decision Rule:

- If $|t^*| \leq t_{crit}$ then do not reject the null hypothesis.
- If $|t^*| > t_{crit}$ then reject the null.

Once a linear model was developed, its specification was assessed. Because of the small number of observations it was not practical to test for heteroscedasticity.

IV.10 Model Specification

A plot of residuals against the independent variable was examined for each model. The nonlinearity of a model was checked by visually inspecting this residual plot for patterns such as the hump in Figure 5, or perhaps a 'U' shape.

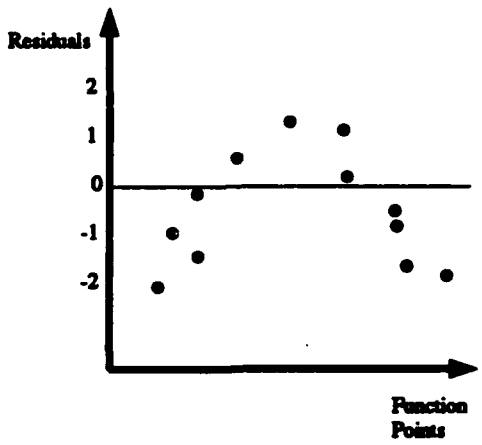


Figure 5. Residual plot against independent variable

IV.11 Distribution of error terms

The assumption of the regression model is that the error terms are normally distributed with a constant variance. With a very small data set it is very difficult to test for these assumptions. The normality of the error terms was evaluated by plotting the residuals from each model as a histogram. If the histogram was reasonably symmetrical, residuals were assumed to be normal.

IV.12 Outliers

With respect to X:

The hat matrix statistic was used to detect outliers with respect to the independent variable. The SAS PROC REG computes the hat matrix, which expresses fitted values as a linear combination of the dependent variables using the elements of the hat matrix $H = X(X'X)^{-1}X'$. The diagonal elements of the hat matrix (h_{ii}) are called leverage values. A leverage value indicates whether the i-th case is distant from the center of all independent variable observations. The leverage is considered large if $h_{ii} > 2 \times \overline{h}$, where $\overline{h} = \frac{p}{n}$, so $h_{ii} > \frac{2p}{n}$ indicates an outlier (21:395–396).

With respect to Y:

The SAS PROC REG provides studentized deleted residuals, which were analyzed to detect outliers with respect to the dependent variable. Studentized deleted residuals with absolute values > $t(\alpha,n-p-1)$, were considered extreme outliers at a level of significance of α=0.05.

IV.13 Influential Outliers

The next step is to ascertain the influence of outlying observations, using the statistic DFFITS, provided by SAS PROC REG. The DF stands for difference, and DFFITS is the difference between:

- the fitted value for the i-th case when all n cases are used in fitting the regression function and
- the fitted value for the i-th case obtained when the i-th case is omitted in fitting the regression function.

It is standardized so that the value $(DFFITS)_i$ for the i-th data point represents roughly the number of estimated standard deviations that the fitted value changes when the i-th case is removed from the data set (21:401):

$$(DFFITS)_i = \frac{Y_i - \hat{Y}_{i(i)}}{\sqrt{MSE_{(i)} h_{ii}}} \tag{14}$$

If the influence statistics exceed established cutoff values, further investigation is warranted. For small data sets, the observation is considered influential if DFFITS > 1. If observations tested as influential, an in-depth analysis was performed to determine if the outlier was the result of measurement error or some other distortion, otherwise a cursory analysis was done.

> When a linear regression model is fitted to a data set with a small number of observations and an outlier is present, the fitted regression may be so distorted by the outlier that the residual plot suggests a lack of fit of the linear regression model in addition to flagging the outlier (21:122–123).

IV.14 Collinearity

Collinearity is a problem arising from two or more independent variables being highly correlated with each other, and are coincidentally explaining the same total sum of the squares. "The overall question when testing for collinearity is: what is the significance of the relations among the predictors?" (27:37) When predictors are correlated, the regression coefficient of any predictor depends on which other predictors are included in the model, and which are left out. Four key questions that can be answered easily if predictors are uncorrelated:

1. What is the relative importance of the effects of different predictor variables ?
2. What is the magnitude of the effect of a given predictor on the response variable ?

3. Can we drop one or more predictors without significantly affecting the explanatory power of the presumed model ?

4. Should any other predictors be included in our model ? (27:37)

Condition numbers, tolerance (TOL), and R^2 were examined for indications of collinearity. There are actually two measures of R^2. The more common one is in the ANOVA table and it is based on the dependent variable Y as a function of the independent variables, or $Y = f(X_1, X_2, X_3, X_4)$, and it will be denoted as $R^2_{Y \cdot X}$. For the purposes of collinearity diagnosis, another coefficient of determination is used which is based on the independent variable in question being a function of the other independent variables, or $X_1 = f(X_2, X_3, X_4)$, and it will be denoted as R^2_X.

Collinearity and tolerance diagnostics examined indicators of correlation among independent variables, using the SAS PROC REG options TOL and COLLINOINT (collinearity with no intercept column in the design matrix). Collinearity exists if a condition number is greater than one. If the condition number > 10, then the model may be adversely affected by collinearity. The TOL option requests the tolerance values for the parameter estimates based on R^2_X. With $TOL = 1 - R^2_X$, a TOL < 0.10 may indicate that collinearity is influencing the least square estimates. The coefficient of determination from the ANOVA table, $R^2_{Y \cdot X}$, was compared against $R^2_X = 1 - TOL$. If $R^2_{Y \cdot X} < R^2_X$, then collinearity was determined to be present.

IV.15 Prediction Intervals

A new observation of work-hours or calendar-weeks corresponding to a given level of function points is viewed as the result of a new trial, independent of the trials on which the regression analysis is based. Prediction of a new observation is subject to:

- variation in possible locations of the distribution of work-hours or calendar-weeks, and
- variation within the probability distribution of work-hours or calendar-weeks (21:81).

Since the true mean response is unknown, with α=0.20 the point estimator $\hat{Y}_h$ will be used to construct a prediction interval:

$$\hat{Y}_h - t_{\alpha, n-p} \times s_{Y_{h(new)}} \leq Y_i \leq \hat{Y}_h + t_{\alpha, n-p} \times s_{Y_{h(new)}} \qquad (15)$$

IV.16 Summary

Data was collected and is displayed in Appendix A along with descriptive statistics from SAS PROCs CHART and REG. The programmers' effort and schedule responses to a given level of support were modeled with indicator variables. After an assessment was made of the model specification, each model was tested. Then a final effort and schedule model was chosen and assessed.

V. Analysis

V.1 Descriptive

The observations listed in Appendix A on page 58 were collected from all personnel available to participate in the research. In addition to the enlisted programmer turnovers, the SCV software support environment transitioned to a new contractor organization during this data collection. The previous support contract was terminated and a new company began supporting the AFIT Oracle baseline at about the time data collection began. As a consequence, Oracle database support activity was below normal throughout the data collection period. The data include Mark II function points supported by a particular Software Enhancement Request and the number of work hours expended by the team towards coding and testing the SER during a period of calendar-weeks. The data was collected per programming team, per project. Twelve observations were collected from AFIT/SCV personnel during the period April through June 1992 and are summarized in Table 9 in Appendix A on page 58. Three of the observations were for teams comprised of two enlisted programmers, and nine were for individuals. Three of the observations were for a single contractor, and the rest were for enlisted programmers. Five were for SQL*Forms support and seven for SQL*Reports. A cursory inspection of the data showed some anomalies in the schedule observations. Team C appeared to have a consistently higher response to a given level of function points than the other teams. The observations are considered accurate because the data collection was carefully conducted. The significant difference in the team C response can be attributed to the fact that team C was interrupted frequently during period of time when the new company was settling into the SCV environment. These unique interruptions are special causes of variation and are not likely to recur. The team C schedule response will be deleted for the purpose of building a schedule model. The *effort* observations for team C will not be deleted because in spite of the interruptions, the applied effort was not unusual.

V.2 Regression

The SAS procedure 'REG' was used to test the models for predicting effort in work-hours and schedule in calendar-weeks based on the number of function points supported. Based on preliminary analysis, an indicator variable was introduced for quantifying the two cases as described in Chapter Four. The indicator variable (I) takes on the value of one for the observations of team D, and zero otherwise.

V.3 Effort Model

V.3.1 Descriptive Statistics

There were 12 observations used to build this model. The effort model describes the relationship between the predictor level of function points and the effort response in work-hours. Most of the observations are at a level of 10 function points or less as shown in the histogram in Appendix A on page 58. The scatter plot on page 60 highlights the preponderance of data at 10 function points or less. There is a generally linear appearance to the relationship between function points and work-hours, but with only two observations greater than 10 function points the possibility of a nonlinear relationship exists. For the purposes of this research a linear relationship is assumed.

V.3.2 Indicator Variables

The values of the indicator variables are:

- I = 0 if team A, B, or C, (single programmers).
- I = 1 if team D (two programmers).

$$\text{MODEL}: WorkHours = 0.538052 + 1.992678 \times ufp - 10.814188 \times I + 2.175083 \times Iufp \quad (16)$$

$$\textbf{Team A, B, C : WorkHours} = 0.538052 + 1.992678 \times \textbf{ufp} \quad (17)$$

$$\textbf{Team D : WorkHours} = -10.276136 + 4.167761 \times \textbf{ufp} \quad (18)$$

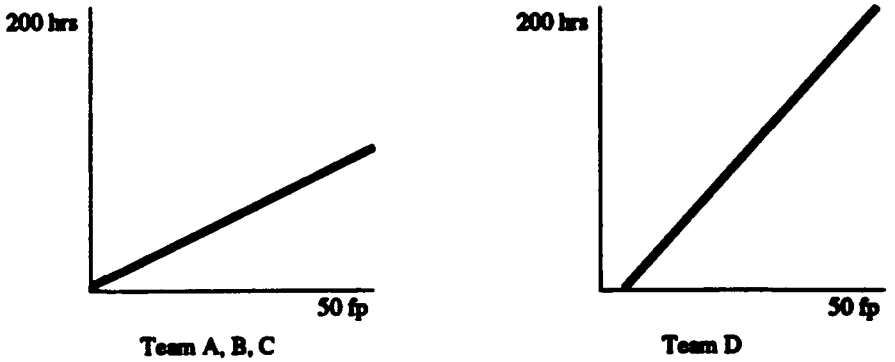


Figure 6. Effort Model Regression Lines

V.3.3 Coefficient of Determination

In Appendix B.1 on page 62 the R^2 value for this model was 0.9759, which indicates that 97.6 percent of the estimating error when the estimate is based on the mean response is explained by the relationship between hours and function points.

V.3.4 F-Ratio

Critical Value: The critical value for the F statistic is $F(\alpha=0.10,3,8)=2.92$ (21:636).

Decision: The F^* value in the ANOVA table in Appendix B is **108.136** which is greater than 2.92, so reject the null hypothesis.

Claim: The probability of obtaining an F-ratio greater than 108.136 if hours was not a function of UFP is less than 0.0001. The relationship appears to be very significant.

V.3.5 Parameter Tests

Critical Value: The critical value for the t statistic is $t(\alpha=0.30,8)=1.108$ (21:630).

Decision: The values of the t-statistics and p-values for the parameters from the ANOVA table, and the decisions for each parameter are summarized in Table 5. The least significant regression coefficient

Table 5

Effort Model Parameter Tests

TERM	PARAMETER	t STATISTIC	p VALUE	DECISION
ufp	β_1	11.688	0.0001	Reject H_o
I	β_2	1.545	0.1609	Reject H_o
Iufp	β_3	5.818	0.0004	Reject H_o

was β_2 which was significant at the 0.16 level of confidence which exceeds the criterion value of 0.30 by a comfortable margin.

V.3.6 Model Specification

Residual plots against the independent variables in Appendix B.2 on page 63 did not reveal any patterns that would indicate a nonlinear relationship.

V.3.7 Distribution of the Error Terms

The error distribution for the effort model in Appendix B.2 shows some skewness to the right. With only 12 observations, however, it is not certain whether or not the error terms are normally distributed. For this model a normal distribution is assumed.

V.3.8 Outliers

The criterion value for identifying outliers with respect to X for this model is $\frac{2p}{n} = \frac{2 \times 4}{12} = 0.75$. From Appendix B.3, observations three and nine had leverage values of **0.97612** and **0.99824** respectively, exceeding the criteria. These observations have the two largest function point values, 50.62 and 32.36 respectively. The next largest observation had 9.8 function points.

The criterion value for identifying outliers with respect to Y using RSTUDENT for this model was $t(\alpha=0.05, 7) = 2.365$ (21:630). Only observation six exceeds this criterion with a value of **2.40519**. Observation six had only 1.92 function points but yet took 17.8 hours to complete This was a large number of hours for the small number of function points. The next largest RSTUDENT was observation three with only 1.4447, which is not a significant value.

V.3.9 Influential Outliers

The outliers examined for influence are three, six, and nine. From Appendix B.3, observations three and nine had DFFITS values much greater than the criterion value of one (**9.2372** and **31.9071** respectively). These were the two observations identified as extreme outliers with respect to X and observations three and nine have the two highest ufp counts. If they were to be removed from the data, the estimating equation would change significantly. Since the software size measurement in function points was exact and any effort measurement error was likely neglible, observations three and nine were not removed for building the model. Observation six, the outlier with respect to Y has a DFFITS value close to one (0.9597), but does not appear to be an influential outlier, and it is a valid although perhaps rare observation. No valid reason was found for removing any of the outliers.

V.3.10 Collinearity

In Appendix B.3, the largest condition number was **2.65931**, which is significantly less than the criterion value of 10. The smallest tolerance value was 0.4371 for IUFP. This means that when regressing IUFP against the remaining independent variables in the model, the R_X^2 would be **0.5629**. This is not a large value, considering the $R_{Y \cdot X}^2$ for the model is 0.9759. Collinearity does not appear to be a problem in this model.

V.3.11 Prediction Intervals

The prediction intervals for teams A, B, C, and D are listed in Appendix B.4 on page 65. Most of the predicted values exceeded the actual values. The prediction for observation nine was equal to the actual value of 125 work-hours. Observation nine at a level of 32.36 function points is predicted to require 125.4 work-hours plus or minus 12 work-hours, 80 percent of the time. For observation three, the effort response to a level of 50.62 function points was predicted to be 101.4 plus or minus 13.9 work-hours, 80 percent of the time. The prediction for the influential observation three was the next best at 101 work-hours compared to the actual 103 work-hours. The next best was observation 11 at 4 work-hours predicted for 1.7 function points with 3.6 work-hours actually observed. Predictions for levels of function points at observations one, five, and 10 were also acceptable. Observation six which happens to have a very large residual value at 13.4 had an actual value outside of the prediction interval.

The relationship between actual effort response and predicted is plotted in Figure 11 on page 65. It shows many close predictions at low levels of function points and the two close predictions out at the influential points.

V.3.12 Closing

In spite of the small data set, the effort model seems to predict rather well up to 50 function points. The indicator variables produce a two-member team model that show an effort response at a rate slightly more than twice that for the one-member model. This seems to imply that there is inefficiency associated with two programmers collaborating on a project.

This effort model can be used to estimate the effort response up to 50 function points. However, this range includes a gap from 10 to 32 function points that is interpolated by the model, with the possibility

that a nonlinear relationship exists in this interpolated region. Additional data collection in this range will reveal the certainty of model.

V.4 Schedule Model One

V.4.1 Descriptive Statistics

After deleting three observations for team C, there were nine observations used to build this model, and are listed in Appendix A on page 58. This schedule model describes the relationship between the predictor level of function points and the schedule response in calendar weeks. Indicator variables were used to differentiate between single and two-programmer efforts.

As with the effort model, most of the function points are at a level less than 10 with two observations above 10. The histogram in Appendix A.3 shows that the distribution of weeks is somewhat different than hours. Of the nine observations there are six that are less than one week and three greater than three weeks. The two highest values of weeks correspond to the two highest function point levels. The second highest level of function points required the highest response at four calendar-weeks. Observation one has a schedule response of three weeks for only 3.58 function points, because the programmer encountered difficulties scheduling testing with the user.

Looking at the scatter plot in Appendix A.3 without observation one for team A at three weeks depicts a relationship similar to the effort model, but with observation one it looks almost nonlinear. For the purpose of this research a linear relationship is assumed.

V.4.2 Indicator Variables

- I = 0 if team A or B (single programmer).
- I = 1 if team D (dual programmers).

$$\text{MODEL: } CalendarWeeks = 0.987992 + 0.038427 \times ufp - 1.142614 \times I + 0.089399 \times Iufp \quad (19)$$

$$\textbf{Team A, B : CalendarWeeks} = 0.987992 + 0.038427 \times \text{ufp} \quad (20)$$

$$\textbf{Team D : CalendarWeeks} = -0.154622 + 0.127826 \times \text{ufp} \quad (21)$$

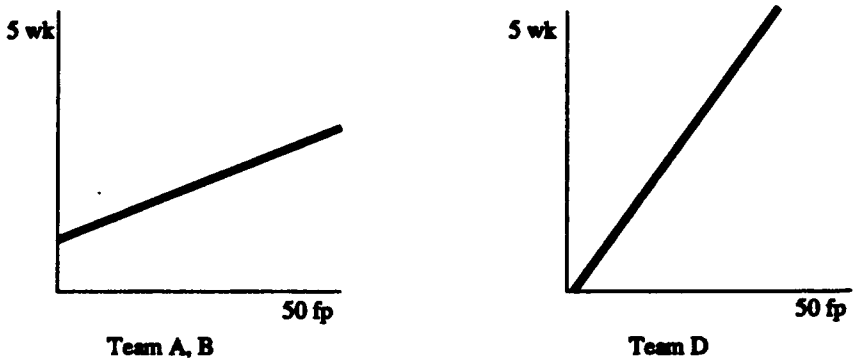


Figure 7. Schedule Model One Regression Lines

V.4.3 Coefficient of Determination

In Appendix C on page 66, the $R^2_{Y \cdot X}$ for this model was **0.6894**, which indicates that 68.9 percent of the estimating error when the estimate is based on the mean response is explained by the relationship between weeks and function points.

V.4.4 F-Ratio

Critical Value: The critical value for the F statistic is $F(\alpha=0.10,3,5)=3.62$ (21:634).

Decision: The F^* value in the ANOVA table in Appendix C is 3.7 which is just greater than 3.62, so reject the null hypothesis.

Claim: The probability of obtaining an F-ratio greater than 3.7 if hours was not a function of UFP is less than **0.0965**. The relationship appears to be significant.

V.4.5 Parameter Tests

Critical Value: The critical value for the t statistic is $t(\alpha=0.30,5)=1.156$ (21:630)

Decision: The values of the t-statistics and p-values for the parameters from the ANOVA table, and the decisions for each parameter are summarized in Table 6. The least significant regression coefficient was

Table 6

Schedule Model One Parameter Tests

TERM	PARAMETER	t STATISTIC	p VALUE	DECISION
ufp	β_1	1.616	0.1671	Reject H_o
I	β_2	1.168	0.2954	Reject H_o
Iufp	β_3	1.788	0.1337	Reject H_o

β_2 which was significant at the 0.295 level of confidence which barely exceeds the criterion value of 0.30.

V.4.6 Model Specification

Residual plots against the independent variables in Appendix C.2 on page 67, did not reveal any patterns that would indicate a nonlinear relationship.

V.4.7 Distribution of the Error Terms

The error distribution plotted on the same page for the schedule model is skewed right, but with only nine observations it is difficult to ascertain the error distribution. For the purposes of this research a normal error distribution is assumed.

V.4.8 Outliers

The criterion value for identifying outliers with respect to X for this model is $\frac{2p}{n} = \frac{2\times4}{9} = 0.89$. In Appendix C.3 on page 68, observations three and nine had leverage values of 0.98297 and 0.99824 respectively, exceeding the criteria. These observations have the two largest function point values, 50.62 and 32.36 respectively. The next largest observation had 9.8 function points.

The criterion value for evaluating outliers with respect to Y using RSTUDENT for this model was $t(\alpha=0.05, 7) = 2.365$. Only observation one exceeds this criterion with a value of 5.47349. Observation one had only 3.58 function points but yet took 3 weeks to complete. This was a large number of weeks for the small number of function points. The next largest RSTUDENT was observation five with only 1.047, which is not a significant value.

V.4.9 Influential Outliers

The outliers examined for influence are one, three, and nine. In Appendix C.3, observations one, three, and nine had DFFITS values much greater than one (2.829, 3.557, and 9.369 respectively). Observations three and nine were identified as extreme outliers with respect to X. If they were to be removed from the data, the estimating equation would change significantly. Observations three and nine have the two highest function point counts. Any effort measurement error was likely neglible and the function point count was exact. Observations three and nine were not removed for building the model. Observation one, the outlier with respect to Y, has a high response to a low level of function points because the programmer encountered difficulties in scheduling testing with the users, so it is valid and will not be removed. No valid reason was found for removing any of the outliers.

V.4.10 Collinearity

In Appendix C.3, the largest condition number was 2.59989, which is significantly less than the criterion value of 10. The smallest tolerance value was 0.4493 for IUFP. This means that when regressing IUFP against the remaining independent variables in the model, the R_X^2 would be 0.5517. Since the $R_{Y \cdot X}^2$ for the model is 0.6894 which is larger than 0.5517, collinearity does not appear to be a problem in this model.

V.4.11 Prediction Intervals

The 80 percent prediction intervals for teams A, B, and D are listed in Appendix C.4 on page 69. About as many of the predicted values were higher than the actual values as they were lower. The prediction for the influential observations three and nine were very close, as well as observation six. The prediction interval for observation three seemed somewhat wide ranging from 1.1 to 4.8 weeks for 50.62 function points. The actual value of observation one, which encountered test schedule difficulties, exceeded the upper bound of the prediction interval. The other observations were in the intervals but all observations except three and nine had negative lower bounds on the prediction interval.

The relationship between actual schedule response and predicted is plotted in Figure 12 on page 69. It shows many close predictions at low levels of function points, and the two influential points are somewhat close to the predicted values.

V.4.12 Closing

Even with a small data set, the schedule model seems to predict fairly well for values of up to 50 function points. The indicator variables produce a two-member team model that show a schedule response at a rate slightly more than three times that for the one-member model. This indicates that there is inefficiency associated with two programmers collaborating on a project.

This model can be used to estimate the schedule response up to 50 function points. However, this range includes a gap from 10 to 32 function points that is interpolated by the model, with the possibility that a nonlinear relationship exists in this interpolated region. Additional data collection in this range will reveal the certainty of model.

Since the 'I' term in Schedule Model One was the least significant at a 0.295 level of confidence, the next model will examine the relationship without the 'I' term.

V.5 Schedule Model 2

V.5.1 Descriptive Statistics

This schedule model describes the relationship between the predictor level of function points and the schedule response in calendar weeks. Only three parameters are estimated in this model, the coefficients of ufp and Iufp, and the intercept. The least significant term, 'I', in Schedule Model One, was dropped for this model.

V.5.2 Indicator Variables

The indicator variables assume the values:

- I = 0 if team A or B (single programmer).
- I = 1 if team D (dual programmers).

$$\text{MODEL} : CalendarWeeks = 0.68226 + 0.046844 \times ufp + 0.049292 \times Iufp \quad (22)$$

$$\textbf{Team A, B : CalendarWeeks} = 0.68226 + 0.046844 \times \text{ufp} \quad (23)$$

$$\textbf{Team D : CalendarWeeks} = 0.68226 + 0.096136 \times \text{ufp} \quad (24)$$

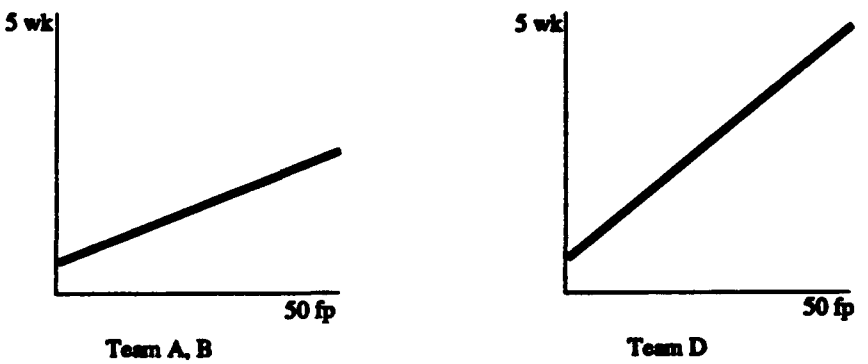


Figure 8. Schedule Model Two Regression Lines

V.5.3 Coefficient of Determination

In Appendix D on page 70, the $R^2_{Y \cdot X}$ for this model was **0.6046**, which indicates that 60.5 percent of the estimating error when the estimate is based on the mean response is explained by the relationship between calendar-weeks and function points, and this is less than Schedule Model One.

V.5.4 F-Ratio

The critical value for the F statistic is $F(\alpha=0.10, 2, 6)=3.46$ (21:634).

Decision: The F^* value in the ANOVA table in Appendix D is **4.588** which is greater than 3.46 so reject the null hypothesis.

Claim: The probability of obtaining an F-ratio greater than 4.588 if weeks was not a function of UFP is less than **0.0618**. The relationship appears to be significant.

V.5.5 Parameter Tests

Critical Value: The critical value for the t statistic is $t(\alpha=0.30,6)=1.134$ (21:630)

Decision: The values of the t-statistics and p-values for the parameters from the ANOVA table, and the decisions for each parameter are summarized in Table 7:

Table 7

Schedule Model Two Parameter Tests

TERM	PARAMETER	t STATISTIC	p VALUE	DECISION
ufp	β_1	2.007	0.0916	Reject H_o
lufp	β_2	1.317	0.2358	Reject H_o

The least significant regression coefficient was β_2 which was significant at the 0.2358 level of confidence which exceeds the criterion value of 0.30.

V.5.6 Model Specification

Residual plots against the independent variables in Appendix D.2 on page 71 did not reveal any patterns that would indicate a nonlinear relationship.

V.5.7 Distribution of the Error Terms

The histogram on the same page shows a right skew due to observation one, which encountered test schedule differences. With only nine observations it is difficult to tell if the errors are normally distributed. A normal distribution is assumed for the purposes of this model.

V.5.8 Outliers

The criterion value for identifying outliers with respect to X for this model would be $\frac{2p}{n} = \frac{2 \times 3}{9} = 0.67$. From Appendix D.3, observations three and nine had leverage values of 0.9725 and 0.9724 respectively, exceeding the criteria. These observations have the two largest function point values, 50.62 and 32.36 respectively.

The criterion value for identifying outliers with respect to Y for this model using RSTUDENT for this model was $t(\alpha=0.05, 5) = 2.571$ (21:630). Only observation one exceeds this criterion. Observation one had 3.58 function points but yet took three weeks to complete. This was a large number of weeks for the small number of function points, due to testing difficulties. The next largest RSTUDENT was only 1.26, not a significant value.

V.5.9 Influential Outliers

The outliers examined for influence are one, three, and nine. Observations one, three, and nine had DFFITS values much greater than one (2.83, 3.56, and 9.37 respectively). Observations three and nine were identified as an extreme outliers with respect to X. If they were to be removed from the data, the estimating equation would change significantly. Observation one, the outlier with respect to Y, is valid and was not removed. No valid reason was found for removing any of the outliers.

V.5.10 Collinearity

In Appendix D.3 the largest condition number was 1.5, which is significantly less than the criterion value of 10. Both tolerance values are 0.85, so if lufp were regressed against ufp, the R_X^2 would be 0.15. Since the $R_{Y \cdot X}^2$ for the model is 0.60 which is much larger than 0.15, collinearity does not appear to be a problem in this model.

V.5.11 Prediction Intervals

The prediction intervals for teams A, B, and D are listed in Appendix D.4 on page 73. Most of the predictions were higher than actual values. The predictions for the influential points three and nine were very close, as was observation two. The actual values for observations four, five, six, and eight were within their predicted intervals, although the lower end of their intervals were negative. The actual value of observation one was 3.0 calendar-weeks and was beyond the intervals upper bound at 2.3 calendar-weeks.

The relationship between actual schedule response and predicted is plotted in Figure 13 on page 73. It shows several close predicted schedule responses at low levels of function points with somewhat worse predictions than Schedule Model One in Figure 12 on page 69 at the influential points.

V.5.12 Closing

The indicator variables produce a two-member team model that show an effort response at a rate slightly more than three times that for the one-member model. This indicates that there is schedule inefficiency associated with two programmers collaborating on a project as opposed to one programmer working on the project when an equal intercept model is used. This second schedule model doesn't seem to predict as well as Schedule Model One for one or two-member teams up to 50 function points. This range includes a gap from 10 to 32 function points that is interpolated by the model, with the possibility

that a nonlinear relationship exists in this interpolated region. Additional data collection in this range will reveal the certainty of model.

V.6 Summary

In this chapter the software support at a given level of function points was related to the effort and schedule to provide that support. The effort model was developed with all 12 observations, but the two schedule models were developed with nine observations. Three schedule observations were deleted because they were a result of special causes of variation. Most of the responses are at a level of 10 function points or less. The two observations at 32 and 50 function points were most influential and resulted in a model that interpolates the responses to requests for software support in the range of 10 to 32 function points.

The use of indicator variables revealed apparent effort and schedule inefficiency when two programmers work on a project as opposed to one. The inefficiency was more pronounced in the schedule models than the effort models. The prediction intervals showed a tendency of the models to overestimate most of the responses to observed levels of function points. The effort model appears to be very significant and predicts reasonably well. Most of the actual values of responses were within the 80 percent prediction interval. Schedule Model One predicts fairly well and better than the second model. Improved schedule estimation will require more data first of all, and most likely additional predictors. The small number of observations prevented a conclusive determination of the distribution of error terms. With the exception of the three schedule observations that were deleted, the remaining observations were subject to typical variations within the population of interest. Schedule Model One will be used to predict schedule for teams A, B, and D because it is a more significant relationship than Schedule Model Two.

VI. Conclusion

A novel technique has been developed for directly estimating the effort and schedule required for Oracle database software support using Mark II function points as a predictor. The characterization of the size of the software support in terms of function points was easily and unambiguously quantified before coding. The relationship between the observed effort and schedule responses and the level of function points was significant and predictable.

VI.1 Measurement Results

VI.1.1 Sizing

The unambiguous mapping of Mark II function points to Oracle components allowed very easy estimation of the size of the software support required before coding. After analysis and design were completed, the programmers were able to exactly quantify how many inputs, outputs, and entities would be added, changed, or deleted during the software support.

VI.1.2 Effort and Schedule

The measurement of effort in work-hours was straightforward. The programmers worked on the projects with a varying number of interruptions. The effort measurements were accurate to one tenth of an hour.

The measurements of schedule in calendar-weeks tended to vary significantly more than the work-hours, due mainly to interruptions from emergent demands upon the programmers. The schedule measurements were accurate to one tenth of a five-day week, or about half of a day.

VI.2 Estimating Results

VI.2.1 Effort

The relationship between a given level of function points and the effort response was found to be very significant, with an apparent inefficiency reflected in the two-person response. The difference is evident in the ufp coefficient for the team D model that is more than twice that for the one-programmer model. The effort model to within two digits of precision is:

$$\text{Team :A, B, C} : WorkHours = 0.54 + 2.0 \times ufp \tag{25}$$

$$\text{Team D}: WorkHours = -10 + 4.2 \times ufp \quad (26)$$

VI.2.2 Schedule

The relationship between a given level of function points and the schedule response was found to be significant, with an apparent inefficiency reflected in the two-person response. The difference is evident in the ufp coefficient for the two-programmer model that is more than three times that for the one-programmer model. The schedule model to within two digits of precision is:

$$\text{Team A, B}: CalendarWeeks = 0.99 + 0.038 \times ufp \quad (27)$$

$$\text{Team D}: CalendarWeeks = -0.15 + 0.13 \times ufp \quad (28)$$

The validity of predictions using these models will depend upon whether basic conditions in the future are similar to those during the period of observation used to build this model. Also, since the model is based on observations ranging up to 50 function points, predictions above that level are less certain. The models built from a small data set and were very dependent upon the influential points. Most observations were at size levels less than 10 function points. The influential points were at much higher levels leaving a gap that is interpolated by the model.

VI.3 Recommendations

VI.3.1 Measurement

Regarding the data collection, a point that needs to be made is that the effort and schedule models predict the work-hours calendar-weeks that will be *reported* as the response to the size level measured in function points. This is not to imply that a programmer would deliberately provide erroneous data, but rather that the programmers are busy supporting the mission, and are likely to spend as little time as possible keeping a log of work-hours spent on a project. As mentioned earlier, the level of support activity was lower than normal, which mitigated any inclination to sacrifice data collection in favor of meeting a schedule and resulted in the excellent quality of data for this research. However, the realities of software support in a demanding environment with tight deadlines will certainly threaten the quality

of data collected, because no matter how much the managers claim that collecting data is of paramount importance, the programmers know that the bottom line is delivering the software.

The measurement of the size of the software support was based on the programmer's completed analysis and design. An alternative would be to use information available after *only* the analysis is complete but not the design. With just the analysis completed, it would be unreasonable to ask a programmer how many trigger-steps will be added, changed, or deleted, but not unreasonable to ask how many table-based blocks will be affected by a software support request. This early count of entities could be used to either estimate the eventual number of inputs and outputs that will be affected, or to directly estimate effort and schedule from the number of additions, changes, and deletions of entities. Also, since only one project had more than one transaction involved, a potential factor could be the number of transactions supported in a project. This count would be especially significant as a predictor of effort and schedule with increased testing due to the interactions between many transactions. The technical complexity adjustment (TCA) was held constant for all of the projects. There may be enough differences within the SCV environment to justify varying the TCA.

The mappings from Oracle components to Mark II function points seem to characterize the functionality very well, but may not be perfect. Specifically, SQL*Forms and SQL*Reports are different software development tools. The assumption is that there is a high correlation between the mappings of each language's components to Mark II function points. With such a small study, it is not feasible to test this assumption. A possibility for further research would be to define many competing mapping strategies, and collect data on all projects using all the mappings. Then for SQL*Forms and SQL*Reports projects that required the same amount of effort or schedule, find out which mapping strategies count them at the same level of function points. This would enhance the usefulness of the weights.

In this research, the size was weighted using Symons' industry average set. The size of the software support could be tabulated by transactions, inputs, entities, and outputs that were added, changed, or deleted as shown in Table 8. A possibility for future research would be to run a multiple regression on the inputs,

Table 8

Multiple Factors

	Additions	Changes	Deletions	TOTAL	SCV Weights
Transactions					W_T
Inputs					W_I
Entities					W_E
Outputs					W_O
TOTAL					
ACD Weights	W_A	W_C	W_D		

entities, and outputs that would provide a set of weights local to SCV. Another possibility would be to perform a multiple regression on the count of weighted additions, changes, and deletions (ACD).

The sizes of the projects in this research were sufficiently small that it was not difficult to estimate the exact number of function points that would be supported. But with larger projects, the difference in estimated and actual function points will become significant. A possible remedy is to have automatic change detection using a before and after function point count. The data collection program in Appendix H currently provides sizing by analogy, and with some modifications could also be used to track changes to the baseline of function points through time. The program could be elevated from the status of a small prototype to an operational sizing and estimating tool using Pro*Ada, an Oracle product that provides database binding to an Ada program (22:50–51). Pro*Ada would allow direct access to the database from the data collection program. Another consideration is to revisit the possibility of using SQL*Reports as the data collection software. The data dictionary report program already accesses the Oracle components. The most convenient aspect of using the data dictionary to count function points is that the Oracle components that are mapped to the function points have names. These names could be viewed as the software baseline, which means that the entire baseline could be viewed in terms of named function points as in Appendix F on page 75.

Currently SCV does not make use of the more advanced capability of SQL*Forms version 3.0 to

imbed procedures into the forms. Should that capability be tapped, the current method of counting function points would likely lose some of the power to explain variation from the effort and schedule mean response to a given level of function points. One possibility would be to include an algorithmic parameter to the three defined by Symons, in the same manner that Jones added a sixth parameter to Albrecht's original five parameters. A fourth parameter added to Mark II function points to measure algorithmic complexity would create a method sufficiently different that it would warrant the designation Mark III.

Since this study was based on relational database software support with a 4GL, it was very easy to adapt Mark II function points to the software support. With a 3GL like COBOL it is not so convenient and would probably entail the accumulation of a lot of rules like Dreger's and Jones', unless the development methodology was focused on inputs, outputs, and entities. This would be easier if a technique known as Hierarchical Input Process Output (HIPO) design was prevalent in the organization, from design, through configuration management, to documentation. Instead of programmers arbitrarily creating modules in an ad hoc manner, but rather designed, documented, and managed the modules as a configuration of Mark II function points, a 3GL project might be as easy to size as the projects in this study. If an organization had both a 3GL and SQL*Forms 3.0, the Mark II function points counting rules they establish for the 3GL could also be applied to any imbedded procedures in SQL*Forms. Any of these suggested changes should be evaluated for their usefulness, and a possibility is using Boehm's 10 criteria for evaluating a software cost model: definition, fidelity, objectivity, constructiveness, detail, stability, scope, ease of use, prospectiveness, and parsimony (6:476).

VI.3.2 Estimating

The single predictor was created by using Symons' Industry Average Set of weights. This limits the potential explanatory power in multiple predictors tabulating the number of additions, changes, and deletions of function points. As a case in point, the two influential observations at 32 and 50 function points appear to indicate a negative relationship with effort when plotted alone as in Figure 9 When the

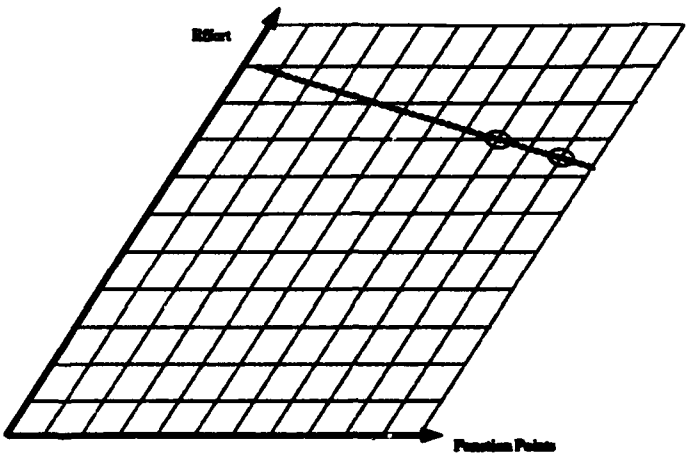


Figure 9. Influential Points

other observations are added along with the regression line as in Figure 10, it shows that the observations are valid although a couple of standard deviations out from the regression line at their levels. Upon closer examination, the observation at 32 function points just happens to be entirely changes, and the one at 50 function points is entirely additions. There is potential for gaining additional explanatory power from the observations by performing a multiple regression based on three factors: additions, changes, and deletions of function points.

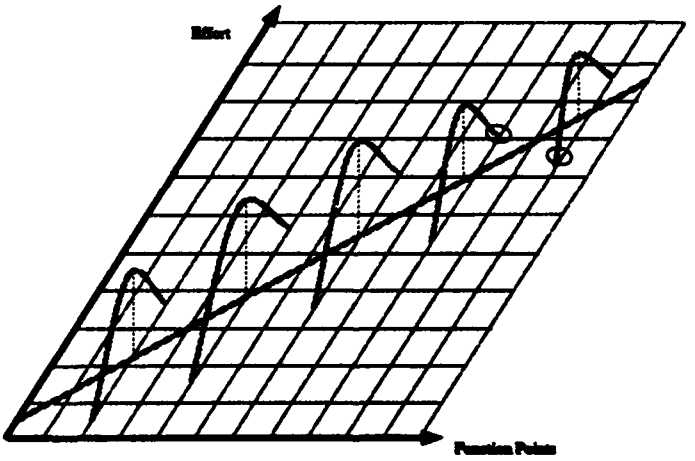


Figure 10. All Observations

Another area lacking data is the response of teams of three, four, or more people. As team members increase arithmetically, the lines of communication among them increase geometrically. It would be

interesting investigate Putnam's assertion that the problem solving response of groups of four or more people are essentially the same (24:216).

Currently computer resource usage by individual is unknown because the accounting information is not activated, and because users and programmers log on with a single group identification account. If accounting information were available by team and project, it could be used to derive date and times to free programmers from schedule data collection.

Humphrey outlined a method of using time series to compensate for a programmer's learning curve and detect when the software development process has reached a stable state (17:203). A problem with his method is that it is based on LOCs and invites its paradoxical effects on software economics. If Humphrey's method were based on a truer measure of software such as function points, the process could be brought under statistical control with a truer sense of continual improvement.

Another benefit of using function points is to study the effect of training. Several months of team observations could be collected, then the training, followed by another run of observations. If the training was effective it might show up as increased efficiency in supporting software requests.

The schedule response is less significantly related to function points than the effort response. This is not surprising since by definition work-hours are only the hours spent working on a particular project, while calendar weeks accumulate for as long as the programmer works on the project, including whatever interruptions delay the project. So, a team's schedule response is likely to be dependent upon other factors in addition to function points. One possibility is defining a factor that reliably characterizes a work order's true priority to the organization. Said in another way, to explain additional variation, define a factor that measures the likelihood of a project being preempted in favor of emergent demands upon the organization. Bob Esterling created an elaborate set of metrics based on the interruptions that retard programmers' productivity (13:164).

There is also a commercial software sizing and estimating product called Before You Leap (BYL) that uses Mark II function points (33:177). It was not available for this research. It may be worth looking into for adaptation to SCV's environment, since it is now clear that there is a significant relationship between Mark II function points and effort and schedule, for the SCV environment. Another possibility is to create a database at AFIT for collecting size, effort, and schedule measurements from many Oracle sites throughout the Air Force. The inclusion of many sites as a study to follow this research would definitely require that the environmental factors be added to the values collected. The benefit would be to create a

software support measurement and estimating system tailored not only to Oracle systems, but to Air Force applications and procedures as well.

VI.4 Software Quality

Many textbooks vainly attempt to define software quality, but there are two books on quality worth adapting to software: Robert Pirsig's classic analysis of quality (23:100) and Dr Deming's idea of operational definitions (11:276–296). Both emphasize a thoughtful approach to quality based on a precise understanding of functionality. Function points can provide a meaningful measure of function-based software quality, that can not only provide a means of sizing and estimating software support required, but can also take advantage of scientific methods of management that have been perfected in other disciplines.

In micro-economics for instance, a firm can minimize its losses by operating at a level where marginal cost is equal to marginal revenue (14:259). This is also related to the concept of complementary slackness in management science, and Barry Boehm described a similar method called marginal analysis (6:209–210). In this research, effort and schedule regression lines were developed, and each line has a slope, which is the marginal cost of supporting function points. The marginal cost in hours or weeks can be converted into dollars. The problem is deriving marginal revenue in a government (or nonprofit) operation.

Revenues can be realized with a recent innovation in the DoD called Defense Business Operating Fund, (DBOF). It is an attempt to enable government organizations to market their products within the government, and actually receive revenue. In fact, one software organization at Gunter AFB, AL is developing software and marketing it within the DoD (9:1). With this measure of revenue, it is possible to derive marginal revenue for function points, allowing the organization to minimize its losses by operating at a level of function points where marginal cost is equal to marginal revenue.

Even without the private sector's convenient common denominator in the dollar, there are ways to achieve continual improvement of software quality based on function points. A simple example might be the number of LOCs per function point. A measure of quality might be the fewer LOCs per function point the better. This of course may encourage poor programming practices that leads to error-prone software. A more sophisticated definition of how to deliver software functions should be based on a precise specification of a programming style guide, as well as a means for scoring the compliance with the style guide.

With the additional information acquired by measuring software functionality in terms of function

points, managers can achieve greater control of the software support process through greater knowledge of the conditioned responses of the process. As many authors have noted, we cannot manage what we cannot measure!

Appendix A. Observations

- A,B: individual enlisted programmers
- C: individual contractor
- D: team of two enlisted programmers (A and B)

Table 9

Summary of Team Observations

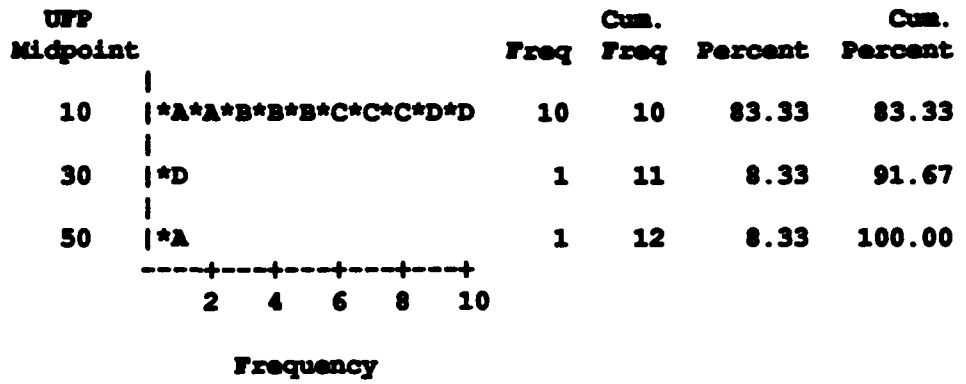
Teams	Government	Contractor	Observations
1 Programmer	6	3	9
2 Programmers	3	0	3
Observation	9	3	12 Total

The SAS System

OBS	HRSCODE	HRSTEST	WKSCODE	WKSTEST	UFP	TEAM	I	HOURS	WEEKS	IUFP
1	4.8	1.0	1.5	1.5	3.58	A	0	5.8	3.0	0.00
2	8.2	3.0	0.6	0.4	9.80	A	0	11.2	1.0	0.00
3	81.0	22.0	2.0	1.0	50.62	A	0	103.0	3.0	0.00
4	4.0	0.9	0.4	0.2	5.24	B	0	4.9	0.6	0.00
5	4.3	0.1	0.1	0.1	3.58	B	0	4.4	0.2	0.00
6	15.8	2.0	0.8	0.2	1.92	B	0	17.8	1.0	0.00
7	4.0	0.5	0.1	0.1	5.24	D	1	4.5	0.2	5.24
8	9.2	2.1	0.4	0.2	3.58	D	1	11.3	0.6	3.58
9	94.0	31.0	2.5	1.5	32.36	D	1	125.0	4.0	32.36
* 10	0.2	0.9	1.5	0.5	1.16	C	0	1.1	2.0	0.00
* 11	1.5	2.1	2.8	0.2	1.74	C	0	3.6	3.0	0.00
* 12	2.1	14.9	2.1	1.9	4.64	C	0	17.0	4.0	0.00

* These observations were used to develop the effort model, but were deleted for the data set used to develop the schedule model.

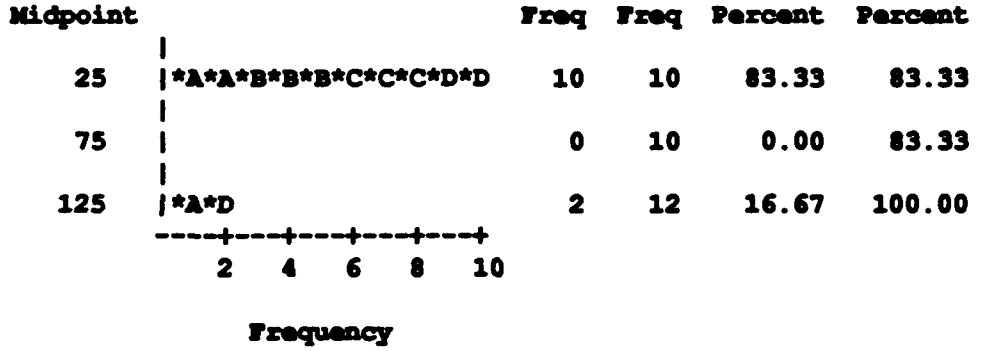
A.1 Frequency Plots



UFP Midpoint		Freq	Cum. Freq	Percent	Cum. Percent
10	*A*A*B*B*B*C*C*C*C*D*D	10	10	83.33	83.33
30	*D	1	11	8.33	91.67
50	*A	1	12	8.33	100.00

2 4 6 8 10

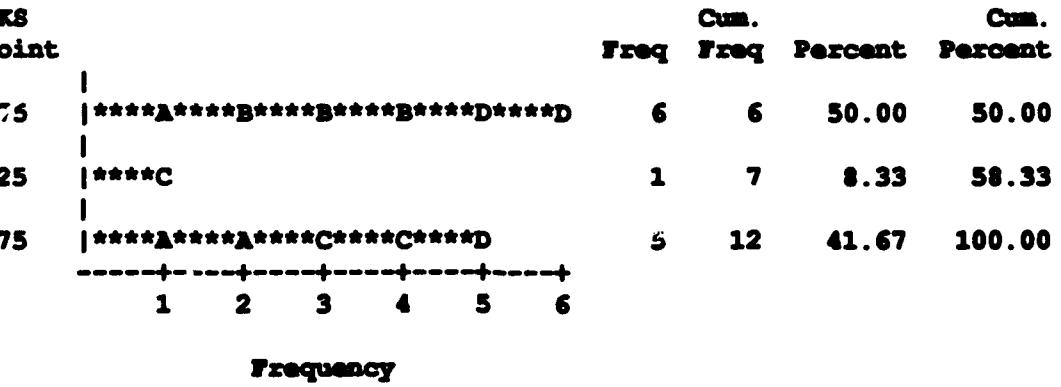
Frequency



HOURS Midpoint		Freq	Cum. Freq	Percent	Cum. Percent
25	*A*A*B*B*B*C*C*C*C*D*D	10	10	83.33	83.33
75		0	10	0.00	83.33
125	*A*D	2	12	16.67	100.00

2 4 6 8 10

Frequency

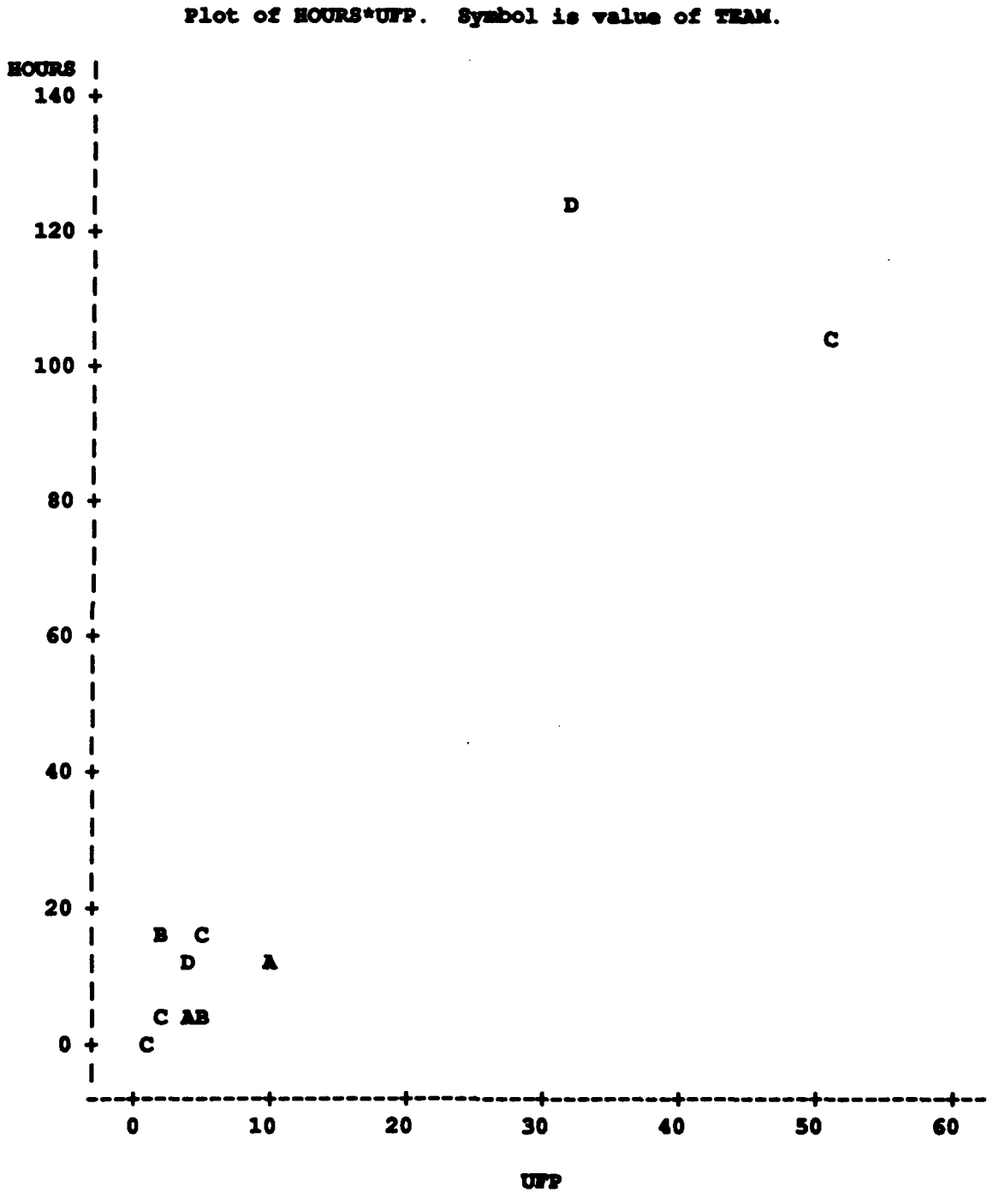


WEEKS Midpoint		Freq	Cum. Freq	Percent	Cum. Percent
0.75	****A****B****B****B****D****D	6	6	50.00	50.00
2.25	****C	1	7	8.33	58.33
3.75	****A****A****C****C****D	5	12	41.67	100.00

1 2 3 4 5 6

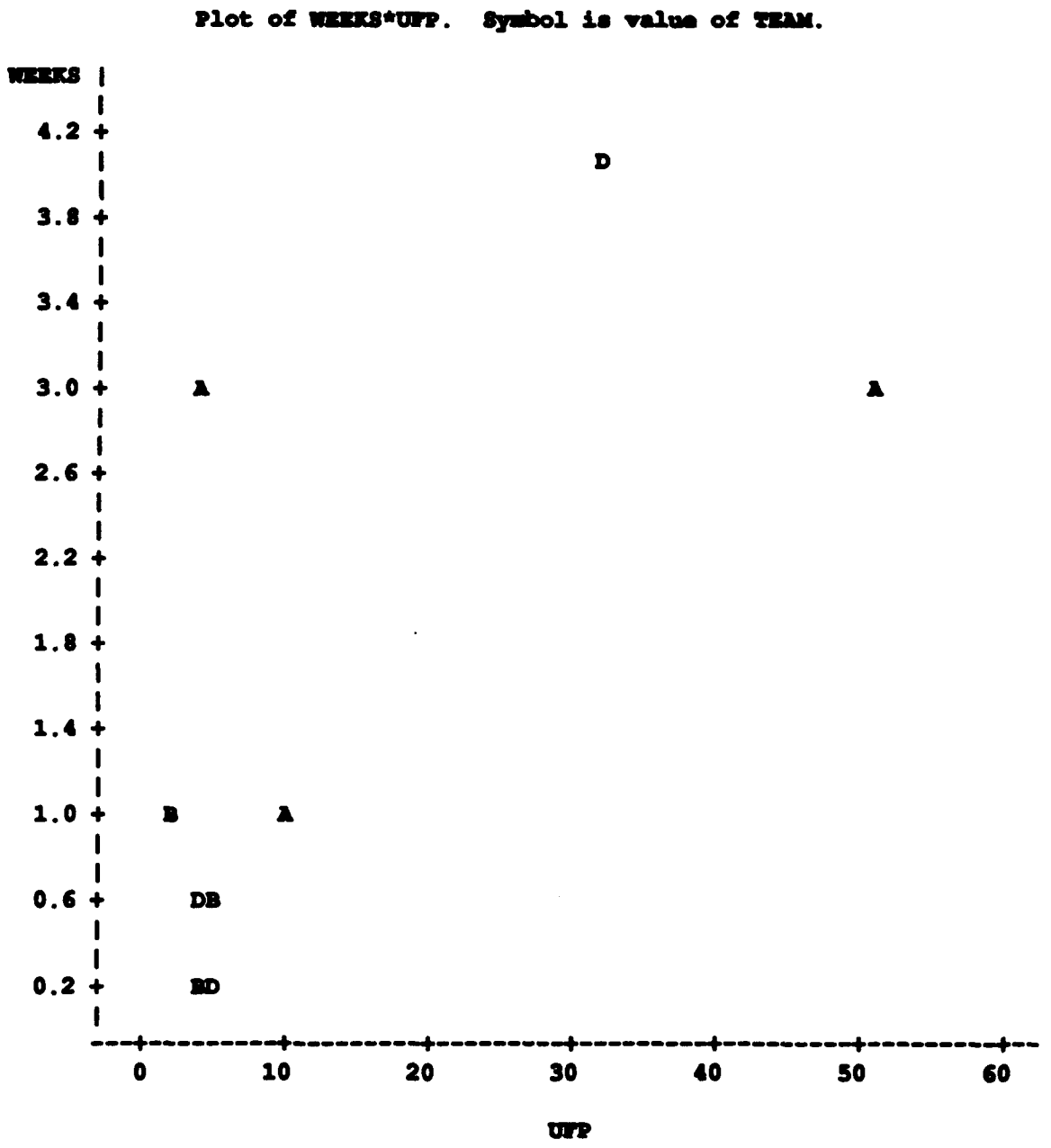
Frequency

A.2 Effort Scatter Plot



NOTE: 2 obs hidden.

A.3 Schedule Scatter Plot



Appendix B. Effort Model

B.1 ANOVA and Parameter Estimates

Model: H
Dependent Variable: HOURS

Analysis of Variance

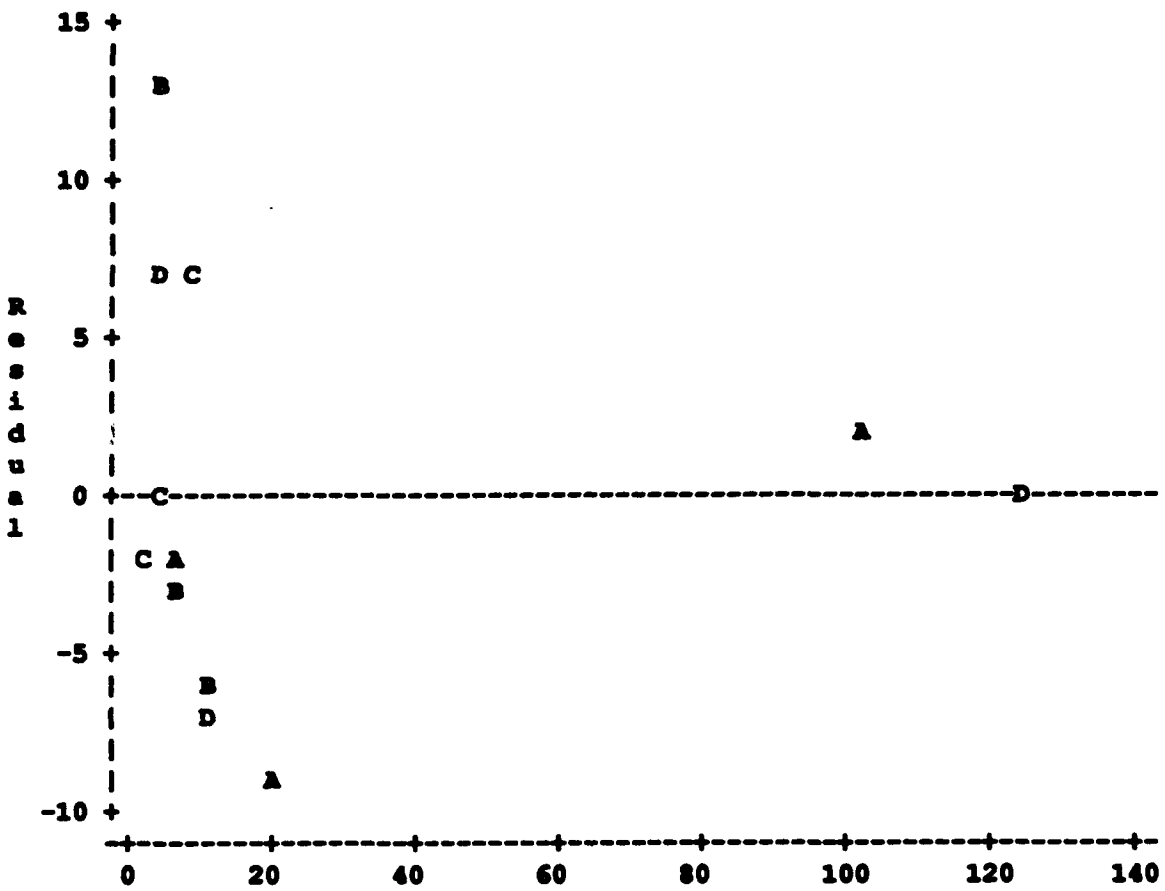
Source	DF	Sum of Squares	Mean Square	F Value	Prob>F
Model	3	18754.23281	6251.41094	108.136	0.0001
Error	8	462.48719	57.81090		
C Total	11	19216.72000			

Root MSE	7.60335	R-square	0.9759
Dep Mean	25.80000	Adj R-sq	0.9669
C.V.	29.47034		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	T for H0: Parameter=0	Prob > |T|
INTERCEP	1	0.538052	2.97537603	0.181	0.8610
UFP	1	1.992678	0.17049057	11.688	0.0001
I	1	-10.814188	6.99881712	-1.545	0.1609
IUFP	1	2.175083	0.37386872	5.818	0.0004

B.2 Model Specification



YRESID Midpoint	Residual	Freq	Cum. Freq	Percent	Cum. Percent
-7.5	****A****B****D	3	3	25.00	25.00
-2.5	****A****B****C****C	4	7	33.33	58.33
2.5	****A****D	2	9	16.67	75.00
7.5	****C****D	2	11	16.67	91.67
12.5	****B	1	12	8.33	100.00

1 2 3 4

Frequency

B.3 Influential Outliers and Collinearity

OBS	HAT	RSTUD	DFITS
1	0.12667	-0.24750	-0.0943
2	0.11133	-1.28667	-0.4554
3	0.97612	1.44470	9.2372
4	0.11877	-0.83556	-0.3067
5	0.12667	-0.43655	-0.1663
6	0.13734	2.40519	0.9597
7	0.47126	-1.33939	-1.2645
8	0.53050	1.33939	1.4237
9	0.99824	1.33939	31.9071
10	0.14315	-0.23343	-0.0954
11	0.13866	-0.05374	-0.0216
12	0.12130	1.01425	0.3768

Variable	DF	Tolerance
INTERCEP	1	.
UFP	1	0.77740809
I	1	0.52453916
IUFP	1	0.43714123

Collinearity Diagnostics(intercept adjusted)

Number	Eigenvalue	Condition Number	Var Prop UFP	Var Prop I	Var Prop IUFP
1	1.85961	1.00000	0.0775	0.1002	0.1080
2	0.87743	1.45581	0.6358	0.1654	0.0029
3	0.26296	2.65931	0.2867	0.7344	0.8891

B.4 Prediction Intervals

Table 10 shows the predicted response, upper, and lower bound of the 80 percent prediction interval. The actual work-hours for each observation is in boldface relative to the predicted values and the bounds.

Table 10

80 Percent Prediction Intervals for Effort Model

OBS	Lower	Actual	Predicted	Actual	Upper	Actual
1	-3.4733	**5.8**	7.6718		18.8170	
2	9.0102	**11.2**	20.0663		31.1224	
3	87.4914		101.4074	**103**	115.3234	
4	-0.1268	**4.9**	10.9797		22.0862	
5	-3.4733	**4.4**	7.6718		18.8170	
6	-6.8305		4.364		15.5585	**17.8**
7	0.4565	**4.5**	11.5629		22.6694	
8	-6.5007		4.6444	**11.3**	15.7896	
9	113.4267	**125**	125.4262		137.4259	
10	-8.3710	**1.1**	2.8496		14.0701	
11	-7.1951	**3.6**	4.0053		15.2058	
12	-1.3351		9.7841	**17**	20.9033	

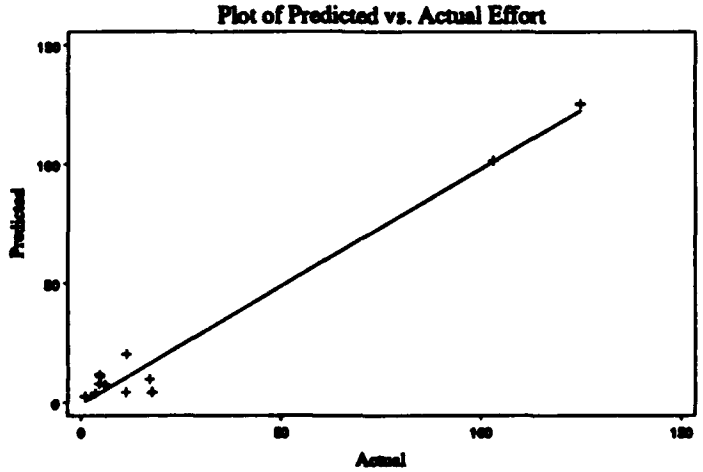


Figure 11. Effort Model Predicted vs Actuals

Appendix C. Schedule Model One

C.1 ANOVA and Parameter Estimates

Model: WF
Dependant Variable: WEEKS

Analysis of Variance

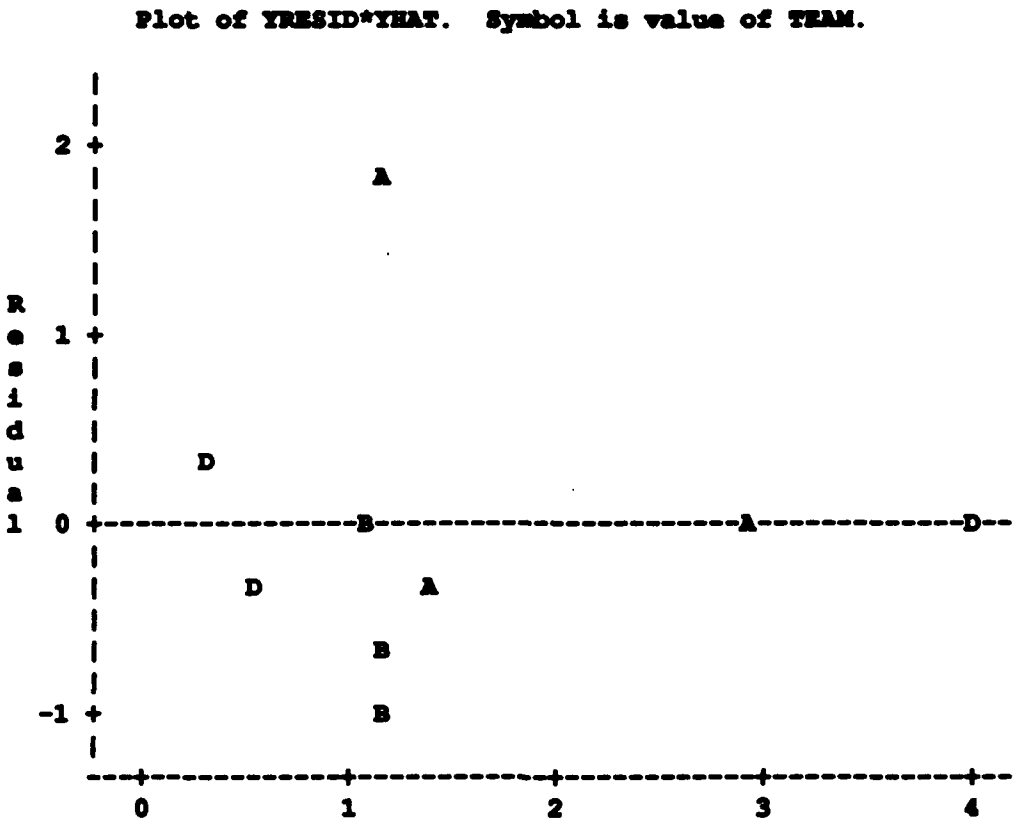
Source	DF	Sum of Squares	Mean Square	F Value	Prob>F
Model	3	11.20229	3.73410	3.700	0.0965
Error	5	5.04660	1.00932		
C Total	8	16.24889			

Root MSE	1.00465	R-square	0.6894
Dep Mean	1.51111	Adj R-sq	0.5031
C.V.	66.48415		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	T for H0: Parameter=0	Prob > |T|
INTERCEP	1	0.987992	0.50596367	1.953	0.1083
UFP	1	0.038427	0.02378450	1.616	0.1671
I	1	-1.142614	0.97807916	-1.168	0.2954
IUFP	1	0.089399	0.04998601	1.788	0.1337

C.2 Model Specification



YRESID Midpoint	Residual	Freq	Cum. Freq	Percent	Cum. Percent
-1.2	****B	1	1	11.11	11.11
-0.4	****A****B****D****D	4	5	44.44	55.56
0.4	****A****B****D	3	8	33.33	88.89
1.2		0	8	0.00	88.89
2.0	****A	1	9	11.11	100.00

1 2 3 4

Frequency

C.3 Influential Outliers and Collinearity

OBS	HAT	RSTUD	DFITS
1	0.21083	5.47349	2.82908
2	0.17062	-0.36220	-0.16428
3	0.98297	0.46822	3.55722
4	0.19586	-0.61188	-0.30197
5	0.21083	-1.04699	-0.54116
6	0.22889	-0.06266	-0.03414
7	0.47126	-0.39329	-0.37130
8	0.53050	0.39329	0.41806
9	0.99824	0.39329	9.36903

Variable	DF	Tolerance
INTERCEP	1	.
UFP	1	0.77251197
I	1	0.52753464
IUFP	1	0.44932603

Collinearity Diagnostics(intercept adjusted)

Number	Eigenvalue	Condition Number	Var Prop UFP	Var Prop I	Var Prop IUFP
1	1.77089	1.00000	0.0630	0.1093	0.1240
2	0.96712	1.35318	0.5892	0.1428	0.0003
3	0.26199	2.59989	0.3478	0.7480	0.8757

C.4 Prediction Intervals

Table 11 shows the predicted response, upper, and lower bound of the 80 percent prediction interval. The actual calendar-weeks for each observation is shown in boldface relative to the predicted values and the bounds.

Table 11

80 Percent Prediction Intervals for Schedule Model One

OBS	Lower	Actual	Predicted	Actual	Upper	Actual
1	-0.3786		1.1256		2.6297	**3.0**
2	-0.1176	**1.0**	1.3646		2.8467	
3	1.0883		2.9332	**3.0**	4.7780	
4	-0.3068	**0.6**	1.1893		2.6855	
5	-0.3786	**0.2**	1.1256		2.6297	
6	-0.4519	**1.0**	1.0618		2.5754	
7	-0.9810	**0.2**	0.5152		2.0113	
8	-1.2011		0.3030	**0.6**	1.8071	
9	2.3968		3.9818	**4.0**	5.5669	

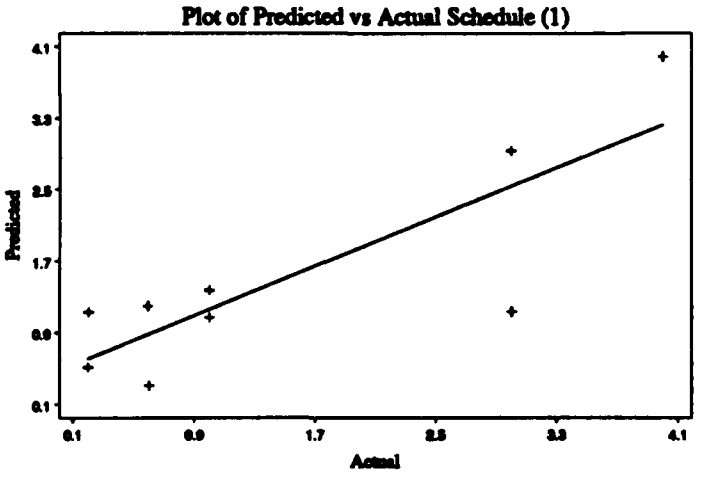


Figure 12. Schedule Model Two Predicted vs. Actual Schedule

Appendix D. Schedule Model Two

D.1 ANOVA and Parameter Estimates

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Prob>F
Model	2	9.82482	4.91241	4.588	0.0618
Error	6	6.42407	1.07068		
C Total	8	16.24889			

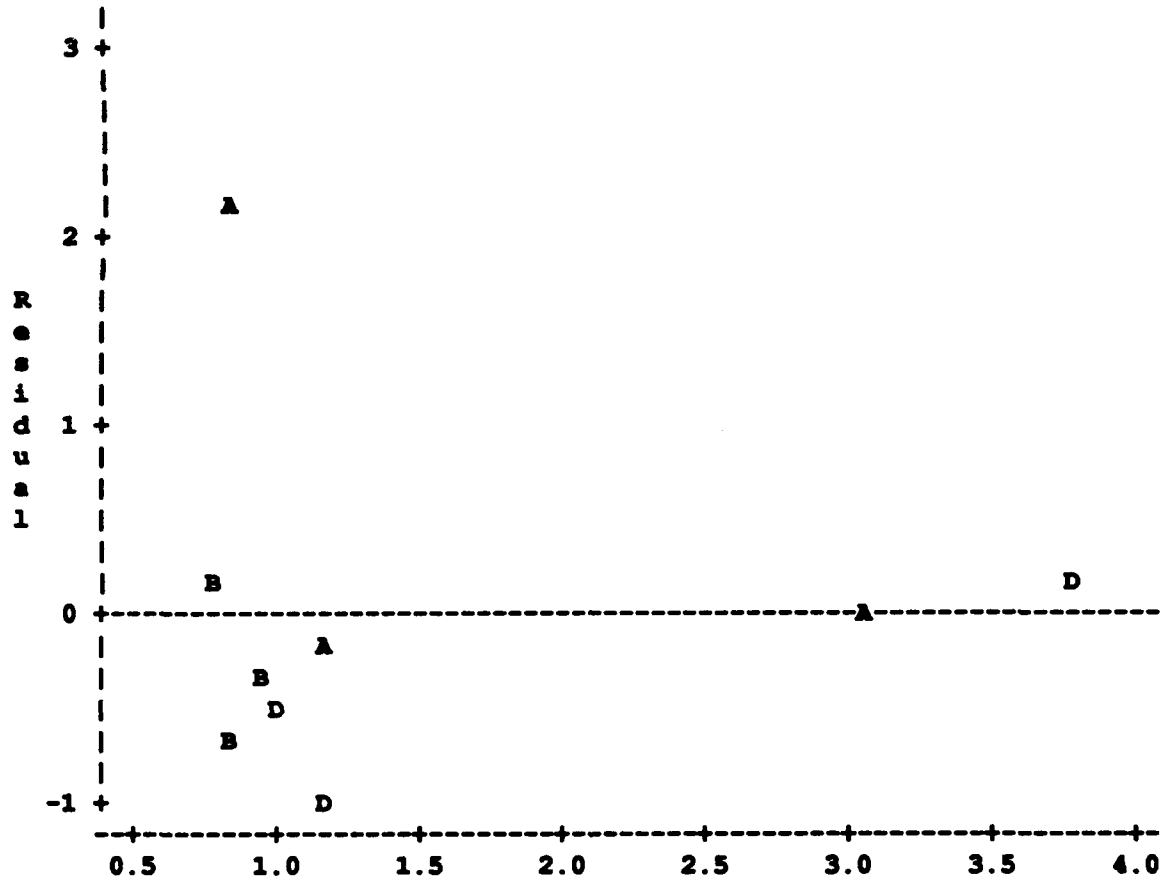
Root MSE	1.03474	R-square	0.6046
Dep Mean	1.51111	Adj R-sq	0.4729
C.V.	68.47514		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	T for H0: Parameter=0	Prob > |T|
INTERCEP	1	0.682226	0.44597194	1.530	0.1770
UFP	1	0.046844	0.02334585	2.007	0.0916
IUFP	1	0.049292	0.03741901	1.317	0.2358

D.2 Model Specification

Plot of YRESID*YHAT. Symbol is value of TEAM.



Predicted Value of WEEKS

YRESID Midpoint	Residual	Freq	Cum. Freq	Percent	Cum. Percent
-1.2	****D	1	1	11.11	11.11
-0.4	****A****A****B****B****D	5	6	55.56	66.67
0.4	****B****D	2	8	22.22	88.89
1.2		0	8	0.00	88.89
2.0	****A	1	9	11.11	100.00

1 2 3 4 5

Frequency

D.3 Influential Outliers and Collinearity

OBS	HAT	RSTUD	DFITS
1	0.21083	5.47349	2.82908
2	0.17062	-0.36220	-0.16428
3	0.98297	0.46822	3.55722
4	0.19586	-0.61188	-0.30197
5	0.21083	-1.04699	-0.54116
6	0.22889	-0.06266	-0.03414
7	0.47126	-0.39329	-0.37130
8	0.53050	0.39329	0.41806
9	0.99824	0.39329	9.36903

Variable	DF	Tolerance
INTERCEP	1	.
UFP	1	0.85055734
IUFP	1	0.85055734

Collinearity Diagnostics(intercept adjusted)

Number	Eigenvalue	Condition Number	Var Prop UFP	Var Prop IUFP
1	1.38658	1.00000	0.3067	0.3067
2	0.61342	1.50346	0.6933	0.6933

D.4 Prediction Intervals

Table 12 shows the predicted response, upper, and lower bound of the 80 percent prediction interval. The actual calendar-weeks for each observation is shown in boldface relative to the predicted values and the bounds.

Table 12

80 Percent Prediction Intervals for Schedule Model Two

OBS	Lower	Actual	Predicted	Actual	Upper	Actual
1	-0.6542		0.8499		2.3541	**3.0**
2	-0.3408	**1.0**	1.1413		2.6234	
3	1.2086	**3.0**	3.0535		4.8983	
4	-0.5685	**0.6**	0.9277		2.4238	
5	-0.6542	**0.2**	0.8499		2.3541	
6	-0.7415		0.7722	**1.0**	2.2858	
7	-0.3102	**0.2**	1.1860		2.6821	
8	-0.4777	**0.6**	1.0264		2.5305	
9	2.2082		3.7932	**4.0**	5.3782	

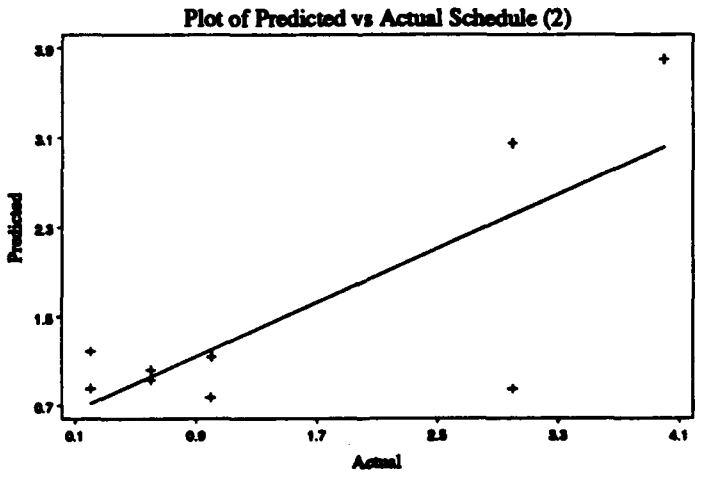


Figure 13. Schedule Model Two Predicted vs Actual

Appendix E. FGREP

E.1 FP2.RE

The text file FP2.RE contains the key strings for the file-oriented regular expression parser, FGREP, to search for in a data dictionary report produced by an Oracle program form_doc.rpt. FGREP is invoked by the Digital Equipment Corp. Command Language file REDUCE.COM in the next section. The contents of FP2.RE are as follows:

```
TITLE:
 TYPE:
STEP =
FIELD
Block
```

E.2 REDUCE.COM

The following DCL file prompts the analyst for information about a particular SER and invokes FGREP to search for Oracle SQL*Forms components that are mapped to Mark II function points as defined in Chapter Three, by the program listed in Appendix H on page 100.

```
$ FileFound = ""
$ FORM = "n"
$ SER  = "n"
$ LOOP2:
$ inquire p2 "What is the SER# (1 to 5 digits) ==> "
$ LOOP1:
$ inquire p1 "Enter file name of .LIS file, without .LIS part ==> "
$ define sys$output 'P1'.fp2
$ fgrep -f fp2.re 'P1'.lis
$ deassign sys$output
$ ren 'P1'.lis 'P1'.'P2'
$ dir/size/date 'P1'*
$ inquire FORM "Do you have another file ? (y/n) "
$ if FORM .eq. "y" then goto LOOP1
$ inquire SER  "Do you have another SER  ? (y/n) "
$ if SER  .eq. "y" then goto LOOP2
$ Done:
```

Appendix F. SQL*Forms Inputs, Entities, Outputs

The following listing is the result of REDUCE.COM scanning a data dictionary report and it represents the portion of the SCV baseline from a SQL*Form (Mark II Transaction) called Admission. Within the listing are the names of the function points. For instance towards the end of the listing, one particular trigger step (Mark II input) can be uniquely distinguished from all the others in the entire baseline by completly specifying its form, block, field, and trigger: ADMISSION, PERSON, SSAN, KEY-PRVFLD, STEP 1.

```
TITLE: ADMISSION
     TYPE: CHECK_MANNING_CODE_TRG
       STEP = 1
     TYPE: CLEAR_DETAILS_TRG
       STEP = 1
     TYPE: KEY-CLRBLK
       STEP = 1
     TYPE: KEY-CREREC
       STEP = 1
     TYPE: KEY-DELREC
       STEP = 1
     TYPE: KEY-DOWN
       STEP = 1
     TYPE: KEY-NXTREC
       STEP = 1
     TYPE: KEY-NXTSET
       STEP = 1
     TYPE: KEY-PRVREC
       STEP = 1
     TYPE: KEY-SCRDOWN
       STEP = 1
     TYPE: KEY-SCRUP
       STEP = 1
     TYPE: KEY-UP
       STEP = 1
     TYPE: PERSON_TRG
       STEP = 1
       STEP = 2
       STEP = 3
     TYPE: PRE-INSERT
       STEP = 1
     TYPE: RESIDENT_TRG
       STEP = 1
       STEP = 2
       Block 1 : PERSON
          TYPE: KEY-CLRREC
            STEP = 1
          TYPE: KEY-ENTQRY
            STEP = 1
          TYPE: KEY-EXEQRY
            STEP = 1
            FIELD 1 : SSAN
            FIELD LENGTH:    11
            TYPE: KEY-PRVFLD
              STEP = 1
                 :
                 :
```

Appendix G. Data Dictionary Program

```
.REM ************************************************************************
.REM This AFIT/SCV RPT application provides hardcopy documentation
.REM for specific SQL*FORMS applications using the Oracle dictionary.
.REM updated 10/90  ME
.REM ************************************************************************

.REM ***** Declare IAPAPP table related variables *****
.REM ***** APPLICATION LEVEL INFORMATION VARIABLES
  .DECLARE APPID 99999        .REM APPLICATION ID NUMBER
  .DECLARE APPOWNER A30       .REM APPLICATION OWNER'S ORACLE USER NAME
  .DECLARE APPNAME A30        .REM SHORT APPLICATION NAME
  .DECLARE APPTITLE A80       .REM TITLE USED FOR MAIN IAP MENU
  .DECLARE TODAY A9           .REM DATE OF REPORT GENERATION


.REM ***** Declare IAPBLK table related variables *****
.REM ***** BLOCK LEVEL INFORMATION VARIABLES
  .DECLARE BLKNAME A30        .REM BLOCK NAME
  .DECLARE BLKDESC A60        .REM MENU LINE DESCRIPTION FOR THIS BLOCK
  .DECLARE BLKSEQ 999         .REM SEQUENCE NUMBER OF BLOCK IN APPL
  .DECLARE BLKUNQKEY A3       .REM Y = CHECK UNIQUENESS OF PRIMARY KEY
  .DECLARE BLKTNAME A61       .REM NAME OF BASE TABLE. NULL-CNTRL BLCK
  .DECLARE TDESCRIPT A60      .REM BASE TABLE DESCRIPTION
  .DECLARE BLKNOREC 99        .REM NUMBER OF ROWS TO DISPLAY
  .DECLARE BLKNOBUF 999       .REM NUMBER OF ROWS TO BUFFER
  .DECLARE BLKBLIN 99         .REM BASE LINE
  .DECLARE BLKLNRC 99         .REM NUMBER OF LINES PER LOGICAL RECORD


.REM ***** Declare all IAPCOMMENT table related variables *****
.REM Primary Key values need not be declared again.
  .DECLARE CMTTEXT A80          .REM COMMENT LINE OF TEXT
  .DECLARE BOILPLATE A80        .REM BOILERPLATE NAME


.REM ***** Declare all IAPFLD table related variables *****
.REM ***** FIELD LEVEL VARIABLES
  .DECLARE FLDNAME A30        .REM FIELD NAME
  .DECLARE FLDSEQ 99          .REM SEQUENCE NUMBER OF FIELD IN BLOCK
  .DECLARE FLDTYPE A7           .REM FIELD DATATYPE
  .DECLARE FLDLEN 999           .REM FIELD LENGTH
  .DECLARE FLDQLEN 999          .REM QUERY LENGTH
  .DECLARE FLDBTAB A3           .REM Y = DATABASE FIELD
  .DECLARE FLDKEY A3            .REM Y = FIELD PART OF PRIMARY
  .DECLARE FLDCKFLD A61         .REM FIELD NAME FROM WHICH TO COPY KEY
  .DECLARE FLDDFLT A80          .REM DEFAULT VALUE
  .DECLARE FLDDISP A3           .REM Y = DISPLAYED FIELD
  .DECLARE FLDPAGE 099          .REM [FLDPAGE] PAGE NUMBER
```

```
  .DECLARE FLDLINE 099          .REM [FLDLINE] LINE NUMBER
  .DECLARE FLDPROMPT A80        .REM FIELD PROMPT
  .DECLARE FLDENTER A3          .REM Y = ENTERABLE FIELD
  .DECLARE FLDQUERY A3          .REM Y = QUERYABLE FIELD
  .DECLARE FLDUPDATE A3         .REM Y = UPDATABLE FIELD
  .DECLARE FLDUPDNUL A3         .REM Y = UPDATEABLE IF NULL FIELD
  .DECLARE FLDMAND A3           .REM Y = MANDATORY FIELD
  .DECLARE FLDFIXED A3          .REM Y = FIXED LENGTH FIELD
  .DECLARE FLDSKIP A3           .REM Y = SKIP TO NEXT FIELD WHEN FULL
  .DECLARE FLDHIDE A3       .REM Y = NO ECHO OF KEYED VALUES TO SCREEN
  .DECLARE FLDAUTOHLP A3    .REM Y = AUTO DISPLAY HELP ON FIELD ENTRY
  .DECLARE FLDUPPER A3          .REM Y = CONVERT TO UPPER CASE
  .DECLARE FLDLOVC A80          .REM NAME OF THE LIST-OF-VALUES COLUMN
  .DECLARE FLDLOW A30           .REM LOW VALUE
  .DECLARE FLDHI A30            .REM HIGH VALUE
  .DECLARE FLDHELP A80          .REM FIELD HELP MESSAGE
  .DECLARE FLDDESCRIPT A60      .REM FIELD DESCRIPTION


.REM ***** Declare IAPTRIGGER related variables *****
.REM ***** TRIGGER VARIABLES
  .DECLARE TRIGTYPE A30     .REM TYPE OF TRIGGER, MACRO
  .DECLARE TRIGDESC A20     .REM TRIGGER DESCRIPTION FOR KEY DISPLAY
  .DECLARE TRIGHIDE A3      .REM Y = DISPLAY TRIGGER IN KEY DISPLA


.REM ***** Declare IAPTRG (Trigger stop) related variables *****
.REM ***** TRIGGER VARIABLES
  .DECLARE TRGSEQ 9999      .REM TRIGGER STEP NUMBER
  .DECLARE TRGLABEL A30     .REM STATEMENT LABEL
  .DECLARE TRGCURS A1       .REM Y = MAINTAIN A SEPARATE CURSOR
  .DECLARE TRGMVE A1        .REM Y = ABORT TRIGGER IF STEP FAILS
  .DECLARE TRGINV A1        .REM Y = REVERSE RETURN CODE
  .DECLARE TRGROLL A1       .REM Y = RETURN FAILURE WHEN ABORTING TRIG
  .DECLARE TRGSLAB A30          .REM SUCCESS LABEL
  .DECLARE TRGFLAB A30          .REM FAILURE LABEL
  .DECLARE TRGMSG A80           .REM MESSAGE DELIVERED ON FAILURE

.REM ***** Declare IAPSQLTXT related variables *****
.REM ***** SQL TEXT VARIABLES
  .DECLARE SQTNO 999        .REM TRIGGER NUMBER ASSIGNED FROM ALL LEVELS
  .DECLARE SQTTEXT A80      .REM TEXT FOR ALL TRIGGERS

.REM ***** Declare IAPMAP table related variables *****
.REM ***** MAP SCREEN VARIABLES
  .DECLARE MAPPAGE 999          .REM BOILERPLATE PAGE NUMBER
  .DECLARE MAPLINE 999          .REM BOILERPLATE LINE NUMBER
  .DECLARE MAPTEXT A80          .REM BOILERPLATE TEXT
  .DECLARE PAGECNT 999          .REM COUNTER FOR BOILERPLATE LOGIC
  .SET PAGECNT 0
  .DECLARE LINECNT 999          .REM COUNTER FOR BOILERPLATE LOGIC
  .DECLARE MAPFLAG A1           .REM 'Y' TURNS ON BOILERPLATE DISPLAY
```

```
  .SET MAPFLAG 'Y'

.REM ***** Other variable declaration *****
  .DECLARE BLOCKS A30 .REM NAME OF BLOCK TO DOCUMENT (OR ALL)


.REM ***** TABLE DEFINITIONS

.REM Main Table - Form Level
#DT 1    5 70 #

.REM Main Table - Block Level
#DT 2    10 70 #

.REM Main Table - Block Level
#DT 3    12 75 #

.REM Main Table - Field Level
#DT 4    15 80 #

.REM Main Table - Field Level
#DT 5    17 90 #

.REM Attributes Table, Form Level
#DT 6    17 32    33 44    45 60    61 80 #

.REM Attributes Form Level
#DT 7    14 14    15 80 #

.REM Fail Message Form Level
#DT 8    15 28    29 80 #

.REM Trigger Step Message Table Block Level
#DT 9    17 0 #

.REM Trigger Step Attributes Table Block Level
#DT 10   19 19    20 80 #

.REM Fail Message Block Level
#DT 11   19 32    33 80 #

.REM Trigger Step Message Field Level
#DT 12   19 0 #

.REM Trigger Step Message Field Level
#DT 13   21 0 #

.REM Trigger Step Attributes Table, Field level
#DT 14   21 21    22 80 #

.REM Fail Message Field level
#DT 15   21 34    35 80 #
```

```
.REM Screen Display Table
#DT 16  1 80 #
#DT 17  17 32   33 80 #

.REM ***** SELECT STATEMENT DEFINITIONS

.REM ***** Define the APPLICATION level report select macro
.DEFINE APPSEL
     SELECT APPID,
            APPTITLE,
            TO_CHAR(SYSDATE)
     INTO   APPID,
            APPTITLE,
            TODAY
     FROM   SYSTEM.IAPAPP
     WHERE  UPPER(APPOWNER)=UPPER(&APPOWNER)
       AND  UPPER(APPNAME)=UPPER(&APPNAME)
..

.REM ***** Define SELECT macro for APPLICATION level comments
.DEFINE APPCMTSEL
     SELECT CMTTEXT
     INTO   CMTTEXT
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  CMTBLK IS NULL
       AND  CMTTRGTYP IS NULL
     ORDER BY CMTLINE
..

.REM ***** Define the SELECT macro for BLOCK level comments
.DEFINE BLKCMTSEL
     SELECT CMTTEXT
     INTO   CMTTEXT
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  CMTBLK = &BLKNAME
       AND  CMTFLD IS NULL
       AND  CMTTRGTYP IS NULL
     ORDER BY CMTLINE
..

.REM ***** Define the SELECT macro for FIELD level comments
.DEFINE FLDCMTSEL
     SELECT CMTTEXT
     INTO   CMTTEXT
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  CMTBLK = &BLKNAME
       AND  CMTFLD = &FLDNAME
       AND  CMTTRGTYP IS NULL
       AND  CMTLINE > 1
```

```
     ORDER BY CMTLINE
..

.REM ***** Define the SELECT macro for FIELD level boilerplates
.DEFINE FLDBOILSEL
     SELECT CMTTEXT
     INTO   BOILPLATE
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  CMTBLK = &BLKNAME
       AND  CMTFLD = &FLDNAME
       AND  CMTTRGTYP IS NULL
       AND  CMTLINE = 1
     ORDER BY CMTLINE
..

.REM ***** SELECT macro for APPLICATION level TRIGGER comments
.DEFINE APPTRIGCMTSEL
     SELECT CMTTEXT
     INTO   CMTTEXT
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  CMTBLK IS NULL
       AND  CMTTRGTYP = &TRIGTYPE
       AND  CMTTRGSEQ = 0
       AND  CMTFLD IS NULL
     ORDER BY CMTLINE
..

.REM ***** SELECT macro for BLOCK level TRIGGER comments
.DEFINE BLKTRIGCMTSEL
     SELECT CMTTEXT
     INTO   CMTTEXT
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  CMTBLK = &BLKNAME
       AND  CMTFLD IS NULL
       AND  CMTTRGTYP = &TRIGTYPE
       AND  CMTTRGSEQ = 0
     ORDER BY CMTLINE
..

.REM ***** SELECT macro for FIELD level TRIGGER comments
.DEFINE FLDTRIGCMTSEL
     SELECT CMTTEXT
     INTO   CMTTEXT
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  CMTBLK = &BLKNAME
       AND  CMTFLD = &FLDNAME
       AND  CMTTRGTYP = &TRIGTYPE
       AND  CMTTRGSEQ = 0
```

```
     ORDER BY CMTLINE
..

.REM ***** Generic SELECT macro for TRIGGER STEP comments
.DEFINE FTRGCMTSEL
     SELECT CMTTEXT
     INTO   CMTTEXT
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  CMTTRGTYP = &TRIGTYPE
       AND  CMTTRGSEQ = &TRGSEQ
       AND  CMTBLK IS NULL
       AND  CMTFLD IS NULL
     ORDER BY CMTLINE
..

.REM ***** Generic SELECT macro for TRIGGER STEP comments
.DEFINE BTRGCMTSEL
     SELECT CMTTEXT
     INTO   CMTTEXT
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  NVL(CMTBLK,'') = &BLKNAME
       AND  CMTTRGTYP = &TRIGTYPE
       AND  CMTTRGSEQ = &TRGSEQ
       AND  CMTFLD IS NULL
     ORDER BY CMTLINE
..

.REM ***** Generic SELECT macro for TRIGGER STEP comments
.DEFINE TRGCMTSEL
     SELECT CMTTEXT
     INTO   CMTTEXT
     FROM   SYSTEM.IAPCOMMENT
     WHERE  CMTAPPID = &APPID
       AND  NVL(CMTBLK,'') = &BLKNAME
       AND  NVL(CMTFLD,'') = &FLDNAME
       AND  CMTTRGTYP = &TRIGTYPE
       AND  CMTTRGSEQ = &TRGSEQ
     ORDER BY CMTLINE
..

.REM ***** Define the SELECT macro for APPLICATION level trig
.DEFINE APPTRIGSEL
     SELECT NVL(TRIGBLK,''),
            NVL(TRIGFLD,''),
            TRIGTYPE,
            NVL(TRIGDESC,'NONE'),
            DECODE(TRIGHIDE,'Y', 'YES','NO')
     INTO   BLKNAME,
            FLDNAME,
            TRIGTYPE,
```

```
            TRIGDESC,
            TRIGHIDE
     FROM   SYSTEM.IAPTRIGGER
     WHERE  TRIGAPPID = &APPID
       AND  TRIGBLK IS NULL
     ORDER BY TRIGTYPE
..

.REM ***** Define the SELECT macro for BLOCK level triggers
.DEFINE BLKTRIGSEL
     SELECT NVL(TRIGFLD,''),
            TRIGTYPE,
            NVL(TRIGDESC,'NONE'),
            DECODE(TRIGHIDE,'Y','YES','NO')
     INTO   FLDNAME,
            TRIGTYPE,
            TRIGDESC,
            TRIGHIDE
     FROM   SYSTEM.IAPTRIGGER
     WHERE  TRIGAPPID = &APPID
       AND  TRIGBLK = &BLKNAME
       AND  TRIGFLD IS NULL
     ORDER BY TRIGTYPE
..

.REM ***** Define the SELECT macro for FIELD level triggers
.DEFINE FLDTRIGSEL
     SELECT TRIGTYPE,
            NVL(TRIGDESC,'NONE'),
            DECODE(TRIGHIDE,'Y','YES','NO')
     INTO   TRIGTYPE,
            TRIGDESC,
            TRIGHIDE
     FROM   SYSTEM.IAPTRIGGER
     WHERE  TRIGAPPID = &APPID
       AND  TRIGBLK = &BLKNAME
       AND  TRIGFLD = &FLDNAME
     ORDER BY TRIGTYPE
..

.REM ***** Define the generic SELECT macro for trigger steps
.DEFINE STEPSEL
     SELECT TRGSEQ,
            TRGLABEL,
            TRGSQL,
            DECODE (TRGCURS,'Y','*',''),
            DECODE(TRGINV,'Y','*',''),
            DECODE (TRGMVE,'Y','*',''),
            DECODE(TRGROLL,'Y','','*'),
            TRGSLAB,
            TRGFLAB,
            TRGMSG
```

```
      INTO    TRGSEQ,
              TRGLABEL,
              SQTNO,
              TRGCURS,
              TRGINV,
              TRGMVE,
              TRGROLL,
              TRGSLAB,
              TRGFLAB,
              TRGMSG
      FROM    SYSTEM.IAPTRG
      WHERE   TRGAPPID = &APPID
        AND   TRGTYPE = &TRIGTYPE
        AND   NVL(TRGBLK,'')=&BLKNAME
        AND   TRGFLD IS NULL
      ORDER BY TRGSEQ
..

.REM ***** Define the FORM level report select macro *****
.DEFINE FSTEPSEL
      SELECT TRGSEQ,
             TRGLABEL,
             TRGSQL,
             DECODE(TRGCURS,'Y','*',''),
             DECODE(TRGINV,'Y','*',''),
             DECODE(TRGMVE,'Y','*',''),
             DECODE(TRGROLL,'Y','','*'),
             TRGSLAB,
             TRGFLAB,
             TRGMSG
      INTO    TRGSEQ,
              TRGLABEL,
              SQTNO,
              TRGCURS,
              TRGINV,
              TRGMVE,
              TRGROLL,
              TRGSLAB,
              TRGFLAB,
              TRGMSG
      FROM    SYSTEM.IAPTRG
      WHERE   TRGAPPID = &APPID
        AND   TRGTYPE = &TRIGTYPE
        AND   TRGBLK IS NULL
        AND   TRGFLD IS NULL
      ORDER BY TRGSEQ
..

.REM ***** Define the FIELD level report trigger select macro
.DEFINE FLDSTEPSEL
      SELECT TRGSEQ,
             TRGLABEL,
```

```
            TRGSQL,
            DECODE(TRGCURS,'Y','*',''),
            DECODE(TRGINV,'Y','*',''),
            DECODE(TRGMVE,'Y','*',''),
            DECODE(TRGROLL,'Y','','*'),
            TRGSLAB,
            TRGFLAB,
            TRGMSG
      INTO  TRGSEQ,
            TRGLABEL,
            SQTNO,
            TRGCURS,
            TRGINV,
            TRGMVE,
            TRGROLL,
            TRGSLAB,
            TRGFLAB,
            TRGMSG
      FROM  SYSTEM.IAPTRG
      WHERE TRGAPPID = &APPID
        AND TRGTYPE = &TRIGTYPE
        AND NVL(TRGBLK,'')=&BLKNAME
        AND NVL(TRGFLD,'')=&FLDNAME
      ORDER BY TRGSEQ
.

.REM ***** Define the generic SELECT macro for trigger text
.DEFINE TRGTXTSEL
      SELECT SQTTEXT
      INTO   SQTTEXT
      FROM   SYSTEM.IAPSQLTXT
      WHERE  SQTAPPID = &APPID
        AND  SQTNO = &SQTNO
      ORDER BY SQTLINE
..

.REM ***** Define the BLOCK level report select macro (ALL BLOCKS)
.DEFINE BLKSEL
      SELECT BLKNAME,
             NVL(BLKDESC,'NONE'),
             BLKSEQ,
             DECODE(BLKUNQKEY,'Y','YES','NO'),
             DECODE(BLKTNAME,NULL,'CONTROL BLOCK',
                    DECODE(BLKTOWNER,NULL,NULL,BLKTOWNER||'.')
                         ||BLKTNAME),
             BLKNOREC,
             BLKNOBUF,
             BLKBLIN,
             BLKLNRC,
             NVL(BLKOBYSQL,0)
      INTO   BLKNAME,
             BLKDESC,
```

```
            BLKSEQ,
            BLKUNQKEY,
            BLKTNAME,
            BLKNOREC,
            BLKNOBUF,
            BLKBLIN,
            BLKLNRC,
            SQTNO
     FROM   SYSTEM.IAPBLK
     WHERE  BLKAPPID = &APPID
     ORDER BY BLKSEQ
..

.REM ***** Define the BLOCK level report select macro (ONE BLOCK)
.DEFINE ONEBLKSEL
     SELECT BLKNAME,
            NVL(BLKDESC,'NONE'),
            BLKSEQ,
            DECODE(BLKUNQKEY,'Y','YES','NO'),
            DECODE(BLKTNAME,NULL,'CONTROL BLOCK',
                   DECODE(BLKTOWNER,NULL,NULL,BLKTOWNER||'.')
                        ||BLKTNAME),
            BLKNOREC,
            BLKNOBUF,
            BLKBLIN,
            BLKLNRC,
            NVL(BLKOBYSQL,0)
     INTO   BLKNAME,
            BLKDESC,
            BLKSEQ,
            BLKUNQKEY,
            BLKTNAME,
            BLKNOREC,
            BLKNOBUF,
            BLKBLIN,
            BLKLNRC,
            SQTNO
     FROM   SYSTEM.IAPBLK
     WHERE  BLKAPPID = &APPID
       AND  UPPER(BLKNAME) = UPPER(&BLOCKS)
..

.REM ***** Define the FIELD level report select macro *****
.DEFINE FLDSEL
     SELECT FLDNAME,
            FLDSEQ,
            FLDTYPE,
            FLDLEN,
            FLDQLEN,
            DECODE(FLDBTAB,'Y','YES','NO'),
            DECODE(FLDKEY,'Y','YES','NO'),
            DECODE(FLDCKBLK,NULL,NULL,FLDCKBLK||'.')||FLDCKFLD,
```

```
            FLDDFLT,
            DECODE(FLDDISP,'Y','YES','NO'),
            DECODE(FLDENTER,'Y','YES','NO'),
            DECODE(FLDQUERY,'Y','YES','NO'),
            DECODE(FLDUPDATE,'Y','YES','NO'),
            DECODE(FLDUPDNUL,'Y','YES','NO'),
            DECODE(FLDMAND,'Y','YES','NO'),
            DECODE(FLDFIXED,'Y','YES','NO'),
            DECODE(FLDSKIP,'Y','YES','NO'),
            DECODE(FLDHIDE,'Y','YES','NO'),
            DECODE(FLDAUTOHLP,'Y','YES','NO'),
            DECODE(FLDUPPER,'Y','YES','NO'),
            DECODE(FLDLOVT,NULL,NULL,FLDLOVT||'.')||FLDLOVC,
            FLDLOW,
            FLDHI,
            FLDHELP,
            FLDPAGE,
            FLDLINE
    INTO    FLDNAME,
            FLDSEQ,
            FLDTYPE,
            FLDLEN,
            FLDQLEN,
            FLDBTAB,
            FLDKEY,
            FLDCKFLD,
            FLDDFLT,
            FLDDISP,
            FLDENTER,
            FLDQUERY,
            FLDUPDATE,
            FLDUPDNUL,
            FLDMAND,
            FLDFIXED,
            FLDSKIP,
            FLDHIDE,
            FLDAUTOHLP,
            FLDUPPER,
            FLDLOVC,
            FLDLOW,
            FLDHI,
            FLDHELP,
            FLDPAGE,
            FLDLINE
    FROM    SYSTEM.IAPFLD
    WHERE   FLDAPPID = &APPID
      AND   FLDBLK = &BLKNAME
    ORDER BY FLDSEQ
..

.REM ***** SELECT macro for FIELD level descrip  *****
.DEFINE FLDDESCSEL
```

```
     SELECT FLDDESCRIPT
     INTO   FLDDESCRIPT
     FROM   FIELD_DESC
     WHERE  CNAME = &FLDNAME
..

.REM ***** SELECT macro for BLOCK level table descrip *****
.DEFINE BLKTDESCSEL
     SELECT TDESCRIPT
     INTO   TDESCRIPT
     FROM   TABLE_DESC
     WHERE  TNAME = &BLKTNAME
..

.REM ***** Define the SELECT macro for BoilerPlate *****
.DEFINE MAPSEL
     SELECT MAPPAGE,
            MAPLINE,
            MAPTEXT
     INTO   MAPPAGE,
            MAPLINE,
            MAPTEXT
     FROM   SYSTEM.IAPMAP
     WHERE  MAPAPPID = &APPID
     ORDER BY MAPPAGE,MAPLINE
..

.REM ***** Define select macro to get logged on
.REM ***** user name (appowner)
.DEFINE GETUSER
     SELECT USER
     INTO   APPOWNER
     FROM   SYSTEM.IAPAPP
..

.REM ***** START RPT

.REM ***** Define the APPLICATION level report body *****
.REM ***** PRINT HEADER, DATE, TITLE, OWNER
.REM ***** RUN FORM, BLOCK, FIELD LEVEL INFO
.DEFINE APPBODY
  #T 1
  #S 2
     ************************************************************
  #S 1
     DOCUMENTATION FOR SQL*FORMS APPLICATION:\ \
.PRINT APPNAME
  #S 1
     ************************************************************
  #S 1
  #RR
     REPORT GENERATION DATE:
```

```
.PRINT TODAY
  #NC
      TITLE:
.PRINT APPTITLE
  #NC
      OWNER:
.PRINT APPOWNER
  #S 2
      ************************************************************
  #CEN  FORM LEVEL
  #N
      ************************************************************
  #N
.REPORT APPCMTSEL CMTBODY CMTHEAD
.TE
.REPORT APPTRIGSEL APPTRIGBODY APPTRIGHEAD
  #T 1
  #S 2
      ************************************************************
  #CEN BLOCK LEVEL
  #N
      ************************************************************
  #TE
.IF "&BLOCKS IS NULL" THEN ALLBLOCKS
.REPORT ONEBLKSEL BLKBODY
.GOTO ENDBLOCKS
.&ALLBLOCKS
.REPORT BLKSEL BLKBODY
.&ENDBLOCKS
.IF "&MAPFLAG != 'Y'" THEN END
    #T 16
    #S 4
.REPORT MAPSEL MAPBODY
    #TE
.&END
..

.REM ***** PRINT COMMENT HEADER
.DEFINE CMTHEAD
    #S 1
        COMMENT:
.CMTBODY
..

.REM ***** PRINT COMMENT TEXT
.DEFINE CMTBODY
.PRINT CMTTEXT
..

.REM ***** FORM LEVEL TRIGGER HEADER
.DEFINE APPTRIGHEAD
    #T 2
```

```
    #S 2
        --------------- TRIGGERS -------------
    #S 1
    #TE
.APPTRIGBODY
..


.REM ***** FORM LEVEL TRIGGER BODY
.DEFINE APPTRIGBODY
      #T 2
          TYPE:
.PRINT TRIGTYPE
      #N
          DESCRIPTION:
.PRINT TRIGDESC
      #N
          HIDE:
.PRINT TRIGHIDE
      #N
      #TE
      #S 1
.REPORT FSTEPSEL STEPBODY
..


.REM ***** FORM LEVEL TRIGGER INFORMATION
.DEFINE STEPBODY
        #T 3
            STEP =
.PRINT TRGSEQ
        #N
            LABEL:
.PRINT TRGLABEL
        #TE
        #T 4
        #S 1
.REPORT TRGTXTSEL TRGTXTBODY
.REPORT FTRGCMTSEL CMTBODY CMTHEAD
        #S 1
        #TE
        #T 7
.PRINT TRGMVE
        #NC
            ABORT TRIGGER WHEN STEP FAILS
        #NC
.PRINT TRGINV
        #NC
            REVERSE RETURN CODE
        #NC
.PRINT TRGROLL
        #NC
```

```
            RETURN SUCCESS ON ABORT
        #NC
.PRINT TRGCURS
        #NC
            SEPARATE CURSOR DATA AREA
        #NC
        #NC
        #S 1
            SUCCESS LABEL:
.PRINT TRGSLAB
        #NC
        #NC
            FAILURE LABEL:
.PRINT TRGFLAB
        #TE
.IFNULL TRGMSG NO_FAIL_MSG
          #T 8
              FAIL MESSAGE:
          #NC
.PRINT TRGMSG
          #TE
.&NO_FAIL_MSG
        #T 4
        #S 1
.REPORT APPTRIGCMTSEL CMTBODY CMTHEAD
        #TE
        #S 2
..

.REM ***** PRINT TRIGGER SQL STATEMENT
.DEFINE TRGTXTBODY
     #RR
.PRINT SQTTEXT
..

.REM ***** BLOCK LEVEL INFORMATION
.DEFINE BLKBODY
    #T 2
    #S 2
        ═══════════════════════════════════════════════
    #N
    \ \ Block
.PRINT BLKSEQ
        :
.PRINT BLKNAME
    #N
        ═══════════════════════════════════════════════
    #S 1
    #TE
    #T 4
    #N
        BLOCK DESCRIPTION:
```

```
.PRINT BLKDESC
    #N
        TABLE NAME:
.PRINT BLKTNAME
    #N
.REPORT BLKTDESCSEL BLKTDESCBODY BLKTDESCHEAD
    #N
.REPORT BLKCMTSEL CMTBODY CMTHEAD
    #S 2
        ---------- ATTRIBUTES ----------
    #N
    #TE
    #T 4
        CHECK FOR UNIQUE KEY:
.PRINT BLKUNQKEY
    #NC
        NUMBER OF ROWS DISPLAYED:
.PRINT BLKNOREC
    #NC
        NUMBER OF ROWS TO BUFFER:
.PRINT BLKNOBUF
    #NC
.IF "&BLKNOREC = 1" THEN LABEL1
          NUMBER OF LINES/ROWS:
.PRINT BLKLNRC
      #NC
          BASE LINE:
.PRINT BLKBLIN
.&LABEL1
    #TE
.IF "&SQTNO = 0" THEN NO_ORDER_BY
      #T 4
      #S 2
        ---------- DEFAULT ORDER BY ----------
      #N
.REPORT TRGTXTSEL TRGTXTBODY
      #TE
.&NO_ORDER_BY
.REPORT BLKTRIGSEL BLKTRIGBODY BLKTRIGHEAD
    #T 3
    #S 1
        **************************************************
    #N
    \ \ \ \ \ FIELDS
    #N
        **************************************************
    #S 1
    #TE
.REPORT FLDSEL FLDBODY
..
```

```
.REM *****
.DEFINE BLKTDESCHEAD
          TABLE DESCRIPTION:
.BLKTDESCBODY
..


.REM *****
.DEFINE BLKTDESCBODY
.PRINT TDESCRIPT
..


.REM ***** BLOCK LEVEL TRIGGER HEADER
.DEFINE BLKTRIGHEAD
      #T 4
      #S 2
          ---------- TRIGGERS ----------
      #S 1
      #TE
.BLKTRIGBODY
..

.REM ***** BLOCK LEVEL TRIGGER BODY
.DEFINE BLKTRIGBODY
        #T 4
        #N
            TYPE:
.PRINT TRIGTYPE
        #N
            DESCRIPTION:
.PRINT TRIGDESC
        #N
            HIDE:
.PRINT TRIGHIDE
        #N
.REPORT BLKTRIGCMTSEL CMTBODY CMTHEAD
        #S 1
        #TE
.REPORT STEPSEL BSTEPBODY
..


.REM ***** BLOCK LEVEL TRIGGER INFORMATION
.DEFINE BSTEPBODY
          #T 9
          #N
              STEP =
.PRINT TRGSEQ
          #N
              LABEL:
.PRINT TRGLABEL
```

```
          #TE
          #T 12
          #S 1
.REPORT TRGTXTSEL TRGTXTBODY
.REPORT BTRGCMTSEL CMTBODY CMTHEAD
          #S 1
          #TE
          #T 10
.PRINT TRGMVE
          #NC
              ABORT TRIGGER WHEN STEP FAILS
          #NC
.PRINT TRGINV
          #NC
              REVERSE RETURN CODE
          #NC
.PRINT TRGROLL
          #NC
              RETURN SUCCESS ON ABORT
          #NC
.PRINT TRGCURS
          #NC
              SEPARATE CURSOR DATA AREA
          #NC
          #NC
          #S 1
              SUCCESS LABEL:
.PRINT TRGSLAB
          #NC
          #NC
              FAILURE LABEL:
.PRINT TRGFLAB
          #NC
          #TE
.IFNULL TRGMSG NO_FAIL_MSG
            #T 11
                FAIL MESSAGE:
            #NC
.PRINT TRGMSG
            #TE
.&NO_FAIL_MSG
          #S 2
..


.REM ***** FIELD LEVEL INFORAMTION
.DEFINE FLDBODY
      #T 4
      #S 2
          **************************************************
      #N
      \ \ FIELD
.PRINT FLDSEQ
```

```
          :
.PRINT FLDNAME
      #N
      #TE
      #T 5
.REPORT FLDDESCSEL FLDDESCBODY FLDDESCHEAD
      #TE
      #T 4
          ************************************************
      #TE
      #T 17
.EXECUTE FLDBOILSEL
      #N
.IF "&FLDPAGE = 0" THEN PAGE_ZERO
             BOILERPLATE:
        #NC
.PRINT BOILPLATE
        #NC
.&PAGE_ZERO
          DATATYPE:
      #NC
.PRINT FLDTYPE
      #NC
          FIELD LENGTH:
      #NC
.PRINT FLDLEN
      #NC
          QUERY LENGTH:
      #NC
.PRINT FLDQLEN
      #NC
          PAGE:
      #NC
.PRINT FLDPAGE
      #NC
          LINE:
      #NC
.PRINT FLDLINE
      #NC
.IFNULL FLDCKFLD NO_FLD_CK
             COPY KEY FROM:
        #NC
.PRINT FLDCKFLD
        #NC
.&NO_FLD_CK
.IFNULL FLDDFLT NO_FLD_DFLT
             DEFAULT VALUE:
        #NC
.PRINT FLDDFLT
        #NC
.&NO_FLD_DFLT
.IFNULL FLDLOW NO_RANGE
```

```
            RANGE LOW:
        #NC
.PRINT FLDLOW
        #NC
            HIGH:
        #NC
.PRINT FLDHI
        #NC
.&NO_RANGE
.IFNULL FLDLOVC NO_FLD_LOVC
          LIST_VAL TABLE:
        #nc
.PRINT FLDLOVC
.&NO_FLD_LOVC
      #NC
.IFNULL FLDHELP NO_FLD_HELP
        #TE
        #T 5
            HELP:
.PRINT FLDHELP
        #TE
        #T 17
.&NO_FLD_HELP
      #TE
      #T 5
.REPORT FLDCMTSEL CMTBODY CMTHEAD
      #S 1
          ---------- ATTRIBUTES ----------
      #S 1
      #TE
      #T 6
          DATABASE FIELD:
      #NC
.PRINT FLDBTAB
      #NC
          PRIMARY KEY:
      #NC
.PRINT FLDKEY
      #NC
          DISPLAYED:
      #NC
.PRINT FLDDISP
      #NC
.IF &FLDDISP='NO' THEN END
            QUERY ALLOWED:
        #NC
.PRINT FLDQUERY
        #NC
            INPUT ALLOWED:
        #NC
.PRINT FLDENTER
        #NC
```

```
.IF &FLDENTER='NO' THEN END
                UPDATE ALLOWED:
            #NC
.PRINT FLDUPDATE
            #NC
                UPDATE IF NULL:
            #NC
.PRINT FLDUPDNUL
            #NC
                FIXED LENGTH:
            #NC
.PRINT FLDFIXED
            #NC
                MANDATORY:
            #NC
.PRINT FLDMAND
            #NC
                AUTOSKIP:
            #NC
.PRINT FLDSKIP
            #NC
                NO ECHO:
            #NC
.PRINT FLDHIDE
            #NC
                AUTO HELP:
            #NC
.PRINT FLDAUTOHLP
            #NC
                UPPER CASE:
            #NC
.PRINT FLDUPPER
            #NC
.&END
      #TE
.REPORT FLDTRIGSEL FLDTRIGBODY FLDTRIGHEAD
..

.REM *****
.DEFINE FLDDESCHEAD
        #RR
            DESCRIPTION:
 .FLDDESCBODY
 ..

.REM *****
.DEFINE FLDDESCBODY
.PRINT FLDDESCRIPT
..


.REM ***** FIELD TRIGGER HEADER
```

```
.DEFINE FLDTRIGHEAD
        #T 4
        #S 2
        \ \ ---------- TRIGGERS ----------
        #S 1
        #TE
.FLDTRIGBODY
..

.REM ***** FIELD LEVEL TRIGGER BODY
.DEFINE FLDTRIGBODY
          #T 5
          #N
              TYPE:
.PRINT TRIGTYPE
          #N
              DESCRIPTION:
.PRINT TRIGDESC
          #N
              HIDE:
.PRINT TRIGHIDE
          #N
.REPORT FLDTRIGCMTSEL CMTBODY CMTHEAD
          #S 1
          #TE
.REPORT FLDSTEPSEL FLDSTEPBODY
..


.REM ***** FIELD LEVEL TRIGGER INFORMATION
.DEFINE FLDSTEPBODY
            #T 12
            #N
                STEP =
.PRINT TRGSEQ
            #N
                LABEL:
.PRINT TRGLABEL
            #TE
            #T 13
            #S 1
.REPORT TRGTXTSEL TRGTXTBODY
.REPORT TRGCMTSEL CMTBODY CMTHEAD
            #S 1
            #TE
            #T 14
.PRINT TRGMVE
            #NC
                ABORT TRIGGER WHEN STEP FAILS
            #NC
.PRINT TRGINV
            #NC
```

```
                REVERSE RETURN CODE
             #NC
.PRINT TRGROLL
             #NC
                RETURN SUCCESS ON ABORT
             #NC
.PRINT TRGCURS
             #NC
                SEPARATE CURSOR DATA AREA
             #NC
             #NC
             #S 1
                SUCCESS LABEL:
.PRINT TRGSLAB
             #NC
             #NC
                FAILURE LABEL:
.PRINT TRGFLAB
             #NC
             #TE
.IFNULL TRGMSG NO_FAIL_MSG
               #T 15
                  FAIL MESSAGE:
               #NC
.PRINT TRGMSG
               #TE
.&NO_FAIL_MSG
             #S 2
..



.REM ***** DRAW THE SCREEN
.DEFINE MAPBODY
.IF "&PAGECNT=&MAPPAGE" THEN LABEL1
      #S 1
          PAGE
.PRINT MAPPAGE
      #NC
.EQUAL PAGECNT MAPPAGE
      #S 1
      .SET LINECNT 1
.&LABEL1
.IF "&LINECNT=&MAPLINE" THE LABEL2
.IF "&LINECNT+1=&MAPLINE" THEN LABEL3
        #S 1
.&LABEL3
.ADD LINECNT LINECNT 1
.GOTO LABEL1
.&LABEL2
    #CL
```

```
.PRINT MAPTEXT
    #
    #NC
..

.REM ***** Logic to identify the application, start report *****
.GOTO GET_USER
.&BAD_APPID
.TELL "COULDN'T FIND APPLICATION, TRY AGAIN OR CTRL-Y TO QUIT"
.&GET_USER
.EXECUTE GETUSER
.ASK "NAME OF THE SQL*FORMS APPLICATION: " APPNAME
.EXECUTE APPSEL
.IF "&APPID IS NULL" THEN BAD_APPID
.ASK "DISPLAY SCREEN MAP? (Y) " MAPFLAG
.SET BLKNAME " "
.SET FLDNAME " "
.PAGE 0 60
.REPORT APPSEL APPBODY
```

Appendix H. Data Collection Program

```
-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- FACILITY:
--      Air Force Institute of Technology, Wright-Patterson AFB OH
-- ABSTRACT:
--      FP2 provides software support measurement for Oracle
--      database applications, by prompting the analyst for
--      additions, changes, and deletions to the baseline.
--      It also provides sizing by analogy by scanning the baseline
--      of SQL*Forms data dictionary reports identified by the
--      analyst as similar to the software support requested.
-- AUTHOR:
--      Capt Steven D. Radnov, AFIT/LSG GSS92D
-- CREATION DATE:
--      Dec 92
-- MODIFICATION HISTORY:
--      {tbs}
--  ----------------------------------------------------------------------
pragma page;
with Text_IO;
with List_Double_Unbounded_Managed; -- Booch Component

procedure FP2 is pragma Optimize(Time);

type Data_type is digits 3;

package Natural_IO is new Text_IO.Integer_IO(natural);
package FloatPt_IO is new Text_IO.Float_IO(Data_type);

-- Actions necessary to support SER to enhance database applications

type SW_Support_type    is  -- s/w support function point actions

    (ADDED,      -- new functionality

    CHANGED,     -- modification of existing functionality

    DELETED,     -- deletion of existing functionality

    TOTAL);      -- total of all software support function points


type SW_Function_type   is  -- Oracle database application

  (TRANSACTIONS,     -- forms, reports

    INPUTS,          -- trigger steps, select statements
```

```
    ENTITIES,        -- blocks, tables

    OUTPUTS);        -- fields, print statements


-- Unadjusted Function Points matrix

type UFP_type is
  array(SW_Support_type,SW_Function_type) of natural;
pragma page;
-- Technical Complexity Adjustment scale

subtype TCA_Scale_type is natural range 0..5;


-- Technical Complexity Adjustment
-- characteristics for array of scale values

type TCA_Characteristics is (
  Data_Communication,
  Distributed_Function,
  Performance,
  Heavily_Used_Configuration,
  Transaction_Rates,
  OnLine_Data_Entry,
  Design_For_End_User_Efficiency,
  Online_Update,
  Complexity_Processing,
  Usable_In_Other_Applications,
  Installation_Ease,
  Operations_Ease,
  Multiple_Sites,
  Facilitate_Change,
  Requirements_Of_Other_Applications,
  Security_Privacy_Auditability,
  User_Training_Needs,
  Direct_Use_By_Third_Parties,
  Documentation,
  Client_Defined_Characteristics);


-- array of characteristic values

type TCA_type is array(TCA_Characteristics) of TCA_scale_type;
pragma page;
-- loop and string slice constants

SER_Num_Length          : constant positive :=  5;
Max_Name_Length         : constant positive := 40;
Max_File_Name_Length    : constant positive := 65;
Max_Line_Length         : constant positive := 80;
```

```
-- String subtypes

subtype SER_Number_type is string(1..SER_Num_Length);
subtype Name_type       is string(1..Max_Name_Length);
subtype File_Name_type  is string(1..Max_File_Name_Length);
subtype Line_type       is string(1..Max_Line_Length);

FP2_Help_Filename : File_Name_type := (others=>' ');
Any_Char          : character;


-- Input function points

type Step_Node_type is
  record
    Name    : Name_type         := (others => ' ');
    Support : SW_Support_Type   := TOTAL;
  end record;
package Step_List is new
  List_Double_Unbounded_Managed(Step_Node_type);


-- Output function points

type Field_Node_type is
  record
    Name    : Name_type         := (others => ' ');
    Support : SW_Support_Type   := TOTAL;
    Steps   : Step_List.List := Step_List.Null_List;
  end record;
package   Field_List  is new
  List_Double_Unbounded_Managed(Field_Node_type);


-- Process function points

type Block_Node_type is
  record
    Name    : Name_type         := (others => ' ');
    Support : SW_Support_Type   := TOTAL;
    Steps   : Step_List.List := Step_List.Null_List;
    Fields  : Field_List.List := Field_List.Null_List;
  end record;
package   Block_List  is new
  List_Double_Unbounded_Managed(Block_Node_type);


-- Transaction types

type App_Node_type is
  record
    Name    : File_Name_type  := (others => ' ');
```

```
    UFP      : UFP_type := (others => (0, 0, 0, 0) );
    Support  : SW_Support_Type := TOTAL;
    Steps    : Step_List.List := Step_List.Null_List;
    Blocks   : Block_List.List := Block_List.Null_List;
  end record;
package   App_List   is new
  List_Double_Unbounded_Managed(App_Node_type);


-- SER information

type Project_Node_type is
  record
    SER_Num : SER_Number_type    := (others => ' ');
    TEAM_ID : Name_type          := (others => ' ');
    Code_Wks : Data_type         := 0.0;
    Test_Wks : Data_type         := 0.0;
    Code_Hrs : Data_type         := 0.0;
    Test_Hrs : Data_type         := 0.0;
    UFPs     : UFP_type := (others => (0, 0, 0, 0) );
    TCA      : TCA_type := (others => 1);
    Apps     : App_List.List     := App_List.Null_List;
  end record;
package   Project_List is new
  List_Double_Unbounded_Managed(Project_Node_type);


Project : Project_List.List;

Project_Option : character;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Prompt user for project SER #
-- ----------------------------------------------------------------------
procedure Get_SER (SER_Num: out SER_Number_type) is

  Buffer : Line_type;
  String_Length : integer;

begin

  SER_Num := (others => ' ');

  Text_IO.New_Line;
  Text_IO.Put("Enter the Software Engineering Request number : ");
  Text_IO.Get_Line(Buffer, String_Length);

  if String_Length > SER_Num_Length

    then
      Text_IO.New_Line;
      Text_IO.Put("SER_Num truncated to -->" &
```

```
                Buffer(1..SER_Num_length) & "<");
      SER_Num(1..SER_Num_Length) := Buffer(1..SER_Num_Length);
      Text_IO.New_Line;

    else
      SER_Num(1..String_Length) := Buffer(1..String_Length);

    end if;

end Get_SER;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Prompt user for the team ...
-- ----------------------------------------------------------------------
procedure Get_Team (Team_ID: out Name_type) is

  Buffer : Line_type;
  String_Length : integer;

begin

  Text_IO.Put
  ("Enter each member of team alphabetically (using e-mail IDs), ");
  Text_IO.New_Line;
  Text_IO.Put("separated by non-alphabetic characters :");
  Text_IO.New_Line;
  Text_IO.Put(">");
  Text_IO.Get_Line(Buffer, String_Length);

  if String_Length > Max_Name_Length

    then
      Text_IO.New_Line;
      Text_IO.Put("Team ID truncated to -->" &
                  Buffer(1..Max_Name_length) & "<");
      Team_ID := Buffer(1..Max_Name_Length);
      Text_IO.New_Line;

    else
      Team_ID(1..String_Length) := Buffer(1..String_Length);

    end if;

end Get_Team;

pragma page;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Clears screen with DEC VT1xx control codes and puts border at top.
-- ----------------------------------------------------------------------
procedure Clear_Screen(With_Line_Of : in character) is
```

```
  CLS : constant string := character'val(27) & "[2J" &
                           character'val(27) & "[H";
begin

  Text_IO.Put(CLS);

  for i in 1 .. 72 loop
    Text_IO.Put(With_Line_Of);
  end loop;

end Clear_Screen;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Gets a character after prompting with string provided.
-- ----------------------------------------------------------------------
function Get_Option(Prompt : in string) return character is
  Char : character := ' ';
begin
  Text_IO.New_Line;
  Text_IO.Put(Prompt);
  Text_IO.Get(Char);
  return Char;
end Get_Option;
-- ----------------------------------------------------------------------
pragma page;

procedure FP2S(Project: in out Project_List.List)  is separate;

procedure FP2E(Project: in out Project_List.List)  is separate;

procedure FP2A(Project: in out Project_List.List)  is separate;

procedure FP2V(Project: in out Project_List.List)  is separate;

--
procedure FP2H(FP2_Help_Filename : string := "fp2.hlp") is separate;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Displays a header for main program
-- ----------------------------------------------------------------------
procedure Header is

begin
  Text_IO.New_Line;
  Text_IO.Put("Software Support  Sizing and Estimating");
  Text_IO.New_Line;
  Text_IO.Put("with Mark II Function Points for Oracle Databases");
  Text_IO.New_Line;
end Header;
pragma page;
```

```
-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Get options for sizing and estimating
-- ----------------------------------------------------------------------
procedure Get_Option(Project_Option: out character) is

  Option : character;

begin

  Text_IO.New_Line;
  Text_IO.Put("(S)ize, (E)stimate, (A)ctual, (Q)uit, (?) : ");
  Text_IO.Get(Option);
  Project_Option := Option;

exception
  when Text_IO.Data_Error =>
       Project_Option := '?';

end Get_Option;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Footer for main program
-- ----------------------------------------------------------------------
procedure Footer is
begin
  Text_IO.New_Line;
  Text_IO.Put("END FP2");
  Text_IO.New_Line;
end Footer;
-----------------------------------------------------------------------
pragma page;


begin  -- FP2

  Clear_Screen(With_Line_Of=>'=');
  Text_IO.Put(" FP2 ");
  FP2_Help_Filename
    (Max_File_Name_Length-7+1..Max_File_Name_Length):="FP2.HLP";

  Header;

  L:loop

    Get_Option(Project_Option);

    case Project_Option is

      when 'S' | 's' => FP2S(Project);  -- Size

      when 'E' | 'e' => FP2E(Project);  -- Estimate
```

```
      when 'A' | 'a' => FP2A(Project);  -- Actual

      when 'Q' | 'q' => FP2V(Project);  -- Save
                        exit L;

      when others => FP2H(FP2_Help_Filename);

    end case;

    Clear_Screen(With_Line_Of=>'=');
    Text_IO.Put(" FP2 ");

  end loop L;

  Footer;

exception
  when Text_IO.End_Error => null;

end FP2;
-------------------------------------------------------------------


-------------------------------------------------------------------

separate (FP2) procedure FP2A(Project: in out Project_List.List) is
  pragma Optimize(Time);

Project_Node : Project_Node_type;
Finished : character := 'N';

procedure FP2AG(Project: in out Project_Node_type) is separate;

procedure FP2AP(Project: in out Project_Node_type) is separate;

pragma page;
-- -----------------------------------------------------------------
begin  -- FP2A

  Clear_Screen(With_Line_Of=>'-');
  Text_IO.Put(" FP2A ");
  Text_IO.Skip_Line;

  Get_SER(Project_Node.SER_Num);
  Get_Team(Project_Node.Team_ID);

  Text_IO.New_Line;
  Text_IO.New_Line;
  Text_IO.Put("For SER#" & Project_Node.SER_Num &
              ", Team:" & Project_Node.Team_ID);

  FP2AG(Project_Node);
  Project_List.Construct
```

```
    (The_Item => Project_Node, And_The_List => Project);
  FP2AP(Project_Node);

end FP2A;
-------------------------------------------------------------------



-- -----------------------------------------------------------------

separate (FP2.FP2A)
  procedure FP2AG(Project: in out Project_Node_type) is
  pragma Optimize(Time);

pragma page;
-- +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Display header for main procedure FP2AG
-- -----------------------------------------------------------------
procedure Header is
begin
  Text_IO.New_Line;
  Text_IO.New_Line;
  Text_IO.Put
        ("Forms, reports, and/or tables " &
         "ADDED, CHANGED, or DELETED to support the SER#" &
         Project.SER_Num);
  Text_IO.New_Line;
  Text_IO.Put
    ("ADDED:  created a completely new form, report, or table");
  Text_IO.New_Line;
  Text_IO.New_Line;
  Text_IO.Put
  ("CHANGED: components within existing form, report, and/or table");
  Text_IO.New_Line;
  Text_IO.Put
    ("i.e. ADD, CHG, DEL blocks, fields, and/or steps IN a form.");
  Text_IO.New_Line;
  Text_IO.New_Line;
  Text_IO.Put
   ("DELETED: completely deleted existing form, report and/or tbl.");
  Text_IO.New_Line;
  Text_IO.New_Line;
  Text_IO.Put("Enter data in this format: 000.0");
  Text_IO.New_Line;
end Header;
-------------------------------------------------------------------
pragma page;
-- +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--  Get Actual Function Points Supported
-- -----------------------------------------------------------------
procedure Get_Actual is

  Any_Char    : character;
```

```
begin  -- Get_Actual

  Get_Code_Wks:
  begin
    Text_IO.Put("Calendar weeks elapsed during coding  > ");
    FloatPt_IO.Get(Project.Code_Wks);
        exception
            when Text_IO.Data_Error =>
                Text_IO.Put("Enter data in this format: 000.0");
                Text_IO.Put(ASCII.BEL);
                Text_IO.New_Line;
                Text_IO.Put("Calendar weeks elapsed during coding>");
                FloatPt_IO.Get(Project.Code_Wks);
  end Get_Code_Wks;

  Get_Test_Wks:
  begin
    Text_IO.Put("Calendar weeks elapsed during testing > ");
    FloatPt_IO.Get(Project.Test_Wks);
        exception
          when Text_IO.Data_Error =>
           Text_IO.Put("Enter calendar weeks in this format: 000.0");
           Text_IO.Put(ASCII.BEL);
           Text_IO.New_Line;
           Text_IO.Put("Calendar weeks elapsed during testing > ");
           FloatPt_IO.Get(Project.Test_Wks);
  end Get_Test_Wks;

  Get_Code_Hrs:
  begin
    Text_IO.Put("Man hours spent on coding > ");
    FloatPt_IO.Get(Project.Code_Hrs);
        exception
            when Text_IO.Data_Error =>
                Text_IO.Put("Enter man hours in this format: 000.0");
                Text_IO.Put(ASCII.BEL);
                Text_IO.New_Line;
                Text_IO.Put("Man hours spent on coding > ");
                FloatPt_IO.Get(Project.Code_Hrs);
  end Get_Code_Hrs;

  Get_Test_Hrs:
  begin
    Text_IO.Put("Man hours spent on testing > ");
    FloatPt_IO.Get(Project.Test_Hrs);
        exception
          when Text_IO.Data_Error =>
           Text_IO.New_Line;
           Text_IO.Put("Enter test man-hours in this format: 000.0");
           Text_IO.Put(ASCII.BEL);
           Text_IO.New_Line;
           Text_IO.Put("Man hours spent on testing > ");
```

```
            FloatPt_IO.Get(Project.Test_Hrs);
    end Get_Test_Hrs;

    Get_Functions:
    for Functions in SW_Function_type loop

      Get_Support:
      for Support in SW_Support_type loop

          Block:
          begin
          Text_IO.Put
              (SW_Function_type'image(Functions) & "," &
               SW_Support_type'image(Support)     & " > ");
          if Support = TOTAL then
              Project.UFPs(TOTAL,Functions) :=
                  Project.UFPs(ADDED,Functions) +
                  Project.UFPs(CHANGED,Functions) +
                  Project.UFPs(DELETED,Functions);
              Natural_IO.Put(Project.UFPs(TOTAL,Functions));
          else
              Natural_IO.Get(Project.UFPs(Support,Functions));
          end if;

          end Block;

      end loop Get_Support;
      Text_IO.New_Line;
      Text_IO.New_Line;

    end loop Get_Functions;

    Any_Char := Get_Option("Any char to continue > ");

end Get_Actual;
-------------------------------------------------------------------
pragma page;

begin  -- FP2AG

    Header;

    Get_Actual;

end FP2AG;
-------------------------------------------------------------------


-- -----------------------------------------------------------------

separate (FP2.FP2A)
procedure FP2AP(Project: in out Project_Mode_type) is
    pragma Optimize(Time);
```

```
Data_File    : Text_IO.File_type;
Col          : Text_IO.Positive_Count := 1;

pragma page;
begin  -- FP2AP

  Text_IO.Create(File=> Data_File,
                 Mode=> Text_IO.Out_file,
                 Name=> "ser" & Project.SER_Num & ".dat");
  Text_IO.Set_Output(File=> Data_File);

  Text_IO.Put(Project.Team_ID);
  Text_IO.Put(Project.SER_Num);
  FloatPt_IO.Put(Project.Code_Wks);
  FloatPt_IO.Put(Project.Test_Wks);
  FloatPt_IO.Put(Project.Code_Hrs);
  FloatPt_IO.Put(Project.Test_Hrs);

  SW_Support:
  for Support in SW_Support_type loop

    SW_Functions:
    for Functions in SW_Function_type loop
        if not (Support = TOTAL)
            then
                Col := Text_IO.Positive_Count( integer(Col) + 5 );
                Text_IO.Set_Col(To=> Col);
                Natural_IO.Put(Project.UFPs(Support,Functions));
            end if;
    end loop SW_Functions;

  end loop SW_Support;

  Text_IO.New_Line;
  Text_IO.Close(Data_File);
  Text_IO.Set_Output(File=> Text_IO.Standard_Output);

  Any_Char := Get_Option("Any char to continue > ");

end FP2AP;
-------------------------------------------------------------------------


-- -----------------------------------------------------------------------
pragma page;
separate (FP2) procedure FP2E(Project: in out Project_List.List) is
  pragma Optimize(Time);

  Project_Mode  : Project_Mode_type;
  Last_Char     : natural := 0;
  SER_Size      : natural := 0;

  procedure FP2ET(TCA : in out TCA_type) is separate;
```

```
begin  -- FP2E

  Clear_Screen(With_Line_Of=>'-');
  Text_IO.Put(" FP2E ");

  if Project_List.Is_Null(Project)

    then
      Text_IO.Put("No projects.");

    else

      case
        Get_Option("Do you want to change default TCAs (Y/N) ? ") is
        when 'Y' | 'y' => FP2ET(Project_Node.TCA);
        when others => null;
      end case;

      Project_Node := Project_List.Head_Of(Project);
      Text_IO.Put("For SER#"      & Project_Node.SER_Num &
                  ", and team: " & Project_Node.Team_ID);
      Text_IO.New_Line;

      for i in reverse 1..SER_Num_Length loop

        if Project_Node.SER_Num(i) /= ' '
          then
            Last_Char := i;
            exit;
          end if;

      end loop;

      for j in SW_Function_type loop

        Text_IO.New_Line;
        Text_IO.Put("The existing baseline has : ");

        for i in reverse SW_Support_type loop

          Natural_IO.Put(Project_Node.UFPs(i,j)) ;
            Text_IO.Put(" -");
          Text_IO.Put(SW_Support_type'image(i) & " ");
          Text_IO.Put(SW_Function_type'image(j));

          if i = TOTAL
            then
              Text_IO.Put
              (", and SER#" & Project_Node.SER_Num & "requires ...");
            elsif j /= TRANSACTIONS
                then
                  SER_Size := SER_Size + ( Project_Node.UFPs(i,j) );
```

```
            end if;

          Text_IO.New_Line;

        end loop;

      end loop;

      Text_IO.New_Line;
      Text_IO.Put
        ("The size of SER#"       & Project_Node.SER_Num & " is :" &
          natural'image(SER_Size) & " Mark II Function Points.");
      Text_IO.New_Line;
      Text_IO.Put("Estimated effort is <> " &
                  "staff-hours, <> " & "calendar-weeks.");
      Text_IO.New_Line;

      Any_Char := Get_Option("Any character to continue ... ");

    end if;

end FP2E;
-------------------------------------------------------------------------


-- -----------------------------------------------------------------------
pragma page;
separate (FP2.FP2E) procedure FP2ET( TCA : in out TCA_type) is
  pragma Optimize(Time);

Count : natural := 0;
Project_Node : Project_node_type;

-- +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--

-- -----------------------------------------------------------------------
procedure Header is
begin
  Text_IO.Put
    ("(value) of Technical Complexity Adjustment characteristics:");
  Text_IO.New_Line;
end;
-------------------------------------------------------------------------

begin

  Header;

  for i in TCA_Characteristics loop

    Count := Count + 1;
    Text_IO.Put(natural'image(Count) & ": (");
```

```
      Text_IO.Put(TCA_Scale_type'image(Project_Mode.TCA(i)) & ") ");
      Text_IO.Put(TCA_Characteristics'image(i));
      Text_IO.New_Line;

    end loop;

end FP2ET;
-------------------------------------------------------------------------

-- ---------------------------------------------------
separate(FP2)
procedure FP2H(FP2_Help_Filename: string := "fp2.hlp") is

      Help_File : Text_IO.File_type;
      Help_Line : Line_type;
      Last_Char : natural;
      Any_Char  : character;

begin

      Clear_Screen(With_Line_Of=>'-');
      Text_IO.Put(" <ENTER>" );

      Text_IO.Open(Help_File,Text_IO.In_File,FP2_Help_Filename);
      Text_IO.Get_Line(Help_File,Help_Line,Last_Char);

      Help:
      while not Text_IO.End_Of_File loop
          Text_IO.New_Line;
          Text_IO.Put(Help_Line(1..Last_Char));
          Text_IO.Get_Line(Help_File,Help_Line,Last_Char);
      end loop Help;
      Text_IO.Close(Help_File);

exception
      when Text_IO.Name_Error =>
          Text_IO.New_Line;
          Text_IO.New_Line;
          Text_IO.Put(FP2_Help_Filename & " not found.");
          Text_IO.New_Line;
          Any_Char := Get_Option("Any character to continue ... ");

      when Text_IO.End_Error =>
          Text_IO.New_Line;
          Any_Char := Get_Option("Any character to continue ... ");

end FP2H;

-- -------------------------------------------------------------------

separate (FP2) procedure FP2S(Project: in out Project_List.List) is
  pragma Optimize(Time);
```

```
Project_Node : Project_Node_type;
App_Node     : App_Node_type;
Current_App : App_List.List := App_List.Null_List;
Apps         : App_List.List := App_List.Null_List;

procedure FP2SG(App   : in out App_List.List) is separate;

procedure FP2SX(App   : in out App_List.List) is separate;

procedure FP2SR(App   : in out App_List.List) is separate;
pragma page;
-- ------------------------------------------------------------------
begin  -- FP2S

  Clear_Screen(With_Line_Of=>'-');
  Text_IO.Put(" FP2S ");
  Text_IO.Skip_Line;

  Get_SER(Project_Node.SER_Num);
  Get_Team(Project_Node.Team_ID);

  Text_IO.New_Line;
  Text_IO.New_Line;
  Text_IO.Put("For SER#" & Project_Node.SER_Num &
              ", Team:" & Project_Node.Team_ID);

  FP2SG(Project_Node.Apps);      -- Get size of project

  Project_List.Construct
    (The_Item => Project_Node, And_The_List => Project);

  Current_App := Project_Node.Apps;

  while not App_List.Is_Null(Current_App) loop

    Extract: declare App_record : App_Node_type;

    begin

      App_record := App_List.Head_Of(Current_App);

      if App_record.UFP(DELETED,TRANSACTIONS) = 0
        then
          FP2SX(Current_App);
        end if;

      Current_App := App_List.Tail_Of(Current_App);

    end Extract;

  end loop;
```

```
   Current_App := Project_Node.Apps;

   while not App_List.Is_Null(Current_App) loop

     FP2SR(Current_App);
     App_Node := App_List.Head_Of(Current_App);

     for i in SW_Support_type loop

       for j in SW_Function_type loop

         Project_Node.UFPs(i,j) :=
         Project_Node.UFPs(i,j) + App_Node.UFP(i,j);

       end loop;

     end loop;

     Current_App := App_List.Tail_Of(Current_App);

   end loop;

   Project_List.Set_Head
     (Of_The_List=>Project,To_The_Item=>Project_Node);

end FP2S;
-------------------------------------------------------------------------



-- -------------------------------------------------------------------
pragma page;
separate (FP2.FP2S) procedure FP2SG(App : in out App_List.List) is
  pragma Optimize(Time);

-- +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Get Apps
-- -------------------------------------------------------------------
procedure Get_App(Support : in     SW_Support_type;
                  App     : in out App_List.List)  is

  App_record     : App_Node_type;
  Buffer         : Line_type := (others => ' ');
  String_Length  : natural := 0;
  All_Apps       : boolean := false;
  File_Name      : File_Name_type  := (others => ' ');

begin  -- Get_App

  Text_IO.Skip_Line;
pragma page;
  Get_Apps:
  loop
```

```
      File_Name := (others => ' ');
      Buffer    := (others => ' ');
      Text_IO.New_Line;
      Text_IO.Put(SW_Support_type'image(Support) & ">");

      All_Apps := true;
      Text_IO.Get_Line(Buffer,String_Length);
      File_Name := Buffer(1..65);

      for i in 1..65 loop
        if File_Name(i) /= ' '
          then
            All_Apps := false;
            exit;
          end if;
        end loop;

      if All_Apps

        then
          Text_IO.New_Line;

        else
          Text_IO.Put(File_Name);
          App_record.Name    := File_Name;
          App_record.Steps   := Step_List.Null_List;
          App_record.Blocks  := Block_List.Null_List;
          App_record.UFP(Support,Transactions) := 1;
          App_record.UFP(TOTAL,Transactions) := 1;
          App_List.Construct
           (The_Item => App_record, And_The_List => App);

        end if;

    exit Get_Apps when All_Apps;

    end loop Get_Apps;

end Get_App;
-------------------------------------------------------------------------
pragma page;

-- +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Explain what to enter for each support type
-- ---------------------------------------------------------------------
procedure Explain(Support : in SW_Support_type) is

begin

  case Support is

    when ADDED   =>
```

```
    Text_IO.New_Line;
    Text_IO.New_Line;
    Text_IO.Put("*NOTE*  Apps to be ADDED will be sized " &
                " by analogy to existing Apps.");
    Text_IO.New_Line;
    Text_IO.Put
      ("Enter the filenames of '.FP2' files of existing Apps ");
    Text_IO.New_Line;
    Text_IO.Put
      ("-similar- to the Apps that you will be developing for SER :");
    Text_IO.New_Line;

      when CHANGED =>

    Text_IO.New_Line;
    Text_IO.Put
      ("Enter the '.FP2' files of existing Apps to be CHANGED :");
    Text_IO.New_Line;

      when DELETED =>

    Text_IO.New_Line;
    Text_IO.Put
     ("Enter the '.FP2' files of existing Apps to be DELETED : ");
    Text_IO.New_Line;

      when others => null;

    end case;

end Explain;
-------------------------------------------------------------------
pragma page;
-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--       Display header for main procedure FP2SG
-- ----------------------------------------------------------------
procedure Header is
begin
    Text_IO.New_Line;
    Text_IO.New_Line;
    Text_IO.Put
     ("IDENTIFY APPS TO BE ADDED, CHANGED, OR DELETED TO SUPPORT SER");
    Text_IO.New_Line;
    Text_IO.New_Line;
    Text_IO.Put("ADDED: means creating a completely new App.");
    Text_IO.New_Line;
    Text_IO.New_Line;
    Text_IO.Put
      ("CHANGED: means that, within an existing App, you need to ");
    Text_IO.New_Line;
    Text_IO.Put(" ADD, CHG, or DEL  some blocks, fields, or steps. ");
    Text_IO.New_Line;
```

```
    Text_IO.New_Line;
    Text_IO.Put("DELETED: means completely deleting an existing App.");
    Text_IO.New_Line;
    Text_IO.New_Line;
    Text_IO.Put("When entering App names, enter one per prompt. ");
    Text_IO.New_Line;
    Text_IO.Put("To exit from the prompting, enter no file name.");
    Text_IO.New_Line;
    Text_IO.New_Line;
    Text_IO.Put
      ("Now choose one or more of the software support actions:");
    Text_IO.New_Line;
  end Header;
  ------------------------------------------------------------------
  pragma page;

  begin  -- FP2SG

    Header;

  -- Get App name and support action

    Get_App_Action:
    loop

      case Get_Option
        ("App to be (A)dded, (C)hanged, (D)eleted, (n)o more : ") is

        when 'A' | 'a' =>
           Explain(ADDED);
           Get_App(ADDED,App);

        when 'C' | 'c' =>
           Explain(CHANGED);
           Get_App(CHANGED,App);

        when 'D' | 'd' =>
           Explain(DELETED);
           Get_App(DELETED,App);

        when others => exit Get_App_Action;

      end case;

    end loop Get_App_Action;

    Text_IO.New_Line;

  end FP2SG;
  ------------------------------------------------------------------

  -- ------------------------------------------------------
  separate (FP2.FP2S) procedure FP2SR(App : in out App_List.List) is
```

```
  pragma Optimize(Time);


App_Node : App_Node_type;


-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Header for FP2 Size Requirement
-- --------------------------------------------------------------------
procedure Header(App_Name : in File_Name_type) is
begin
  Clear_Screen('-');
  Text_IO.Put(" FP2SR ");
  Text_IO.New_Line;
  Text_IO.Put("Size of the SER for App: " & App_Name);
  Text_IO.New_Line;
end;
--------------------------------------------------------------------------
pragma page;

-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Prompter for FP2 Size Requirement
-- --------------------------------------------------------------------
procedure FP_Changed(UFP : in out UFP_type) is

    Remaining : natural := 0;

    function Prompt_With
      (Question: string; Limit: natural) return natural is

      Function_Points : natural := 0;

    begin

      Get_FP:
      loop

        Block:
        begin

          Text_IO.New_Line;
          Text_IO.Put(Question);
          Natural_IO.Get(Function_Points);

          if Function_Points > Limit
            then
              Text_IO.Put(natural'image(Function_Points) & " >"  &
                          natural'image(Limit) & " remaining,");
            else
              exit Get_FP;
            end if;
```

```
        exception
          when Text_IO.Data_Error =>
            Text_IO.Put("ENTER AN INTEGER");

        end Block;

      end loop Get_FP;

    return Function_Points;

    end Prompt_With;
pragma page;
-------------------------------------------------------------------------
begin  -- FP_Changed

  Support:
  for Supported in ADDED .. DELETED loop

    Functions:
    for Functionality in INPUTS .. OUTPUTS loop

      case Supported is

        when ADDED   =>
          Remaining := natural'LAST;

        when CHANGED =>
          Remaining := natural( UFP(TOTAL,Functionality) );

        when DELETED =>
          Remaining := natural( UFP(TOTAL,Functionality)   ) -
                       natural( UFP(CHANGED,Functionality) );
      end case;

      case Functionality is

        when INPUTS    => UFP(Supported,Functionality)  :=
          Prompt_With("How many trigger  STEPS   will be " &
            SW_Support_type'image(Supported) & ": ", Remaining);

        when ENTITIES  => UFP(Supported,Functionality) :=
          Prompt_With("How many  BLOCKS  will be " &
            SW_Support_type'image(Supported) & ": ", Remaining);

        when OUTPUTS   => UFP(Supported,Functionality) :=
          Prompt_With("How many  FIELDS  will be " &
            SW_Support_type'image(Supported) & ": ", Remaining);

        when others => null;

      end case;
```

```
    end loop Functions;

    Text_IO.New_Line;

  end loop Support;

end FP_Changed;
pragma page;
-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Footer for FP2 Size Requirement
-- ----------------------------------------------------------------------
procedure Footer(App_Name : in File_Name_type) is
begin
  Text_IO.New_Line;
  Text_IO.Put("Sized Requirement for App: " & App_Name);
  Text_IO.New_Line;
end;
-----------------------------------------------------------------------

begin

  if not App_List.Is_Null(App)

    then
      App_Node := App_List.Head_Of(App);

      if App_Node.UFP(CHANGED,TRANSACTIONS) > 0
        then
          Header(App_Node.Name);
          FP_Changed(App_Node.UFP);
          App_List.Set_Head(Of_The_List=>App, To_The_Item=>App_Node);
          Footer(App_Node.Name);
        elsif App_Node.UFP(ADDED,TRANSACTIONS) > 0
          then
            for i in INPUTS .. OUTPUTS loop
              App_Node.UFP(ADDED,i) := App_Node.UFP(TOTAL,i);
            end loop;
          App_List.Set_Head(Of_The_List=>App, To_The_Item=>App_Node);
        end if;

    else
      Text_IO.Put("Empty App list");

    end if;

end FP2SR;
-----------------------------------------------------------------------


-- ----------------------------------------------------------------------
with Pattern_Match_Regular_Expression; -- Booch Component

  package GREP   is new Pattern_Match_Regular_Expression
```

```
    (Item          => character,
     Index         => positive.
     Items         => string,
     Any_Item      => '?',
     Escape_Item   => '\',
     Not_Item      => '^',
     Closure_Item  => '*',
     Start_Class   => '[',
     Stop_Class    => ']',
     Is_Equal      => "=");
pragma page;
-- ------------------------------------------------------------------------

with GREP;
separate (FP2.FP2S) procedure FP2SX (App: in out App_List.List) is
  pragma Optimize(Time);


  type Part_type  is

     (NULL_PART,          -- ignore, continue scanning

     BLOCK_PART,          -- block detected

     FIELD_PART,          -- field detected

     TRIGGER_NAME_PART, -- trigger name detected, wait for step num

     STEP_NUMBER_PART); -- step number, trigger step ID complete



  type Level_type is

  ( APP_LEVEL,       -- looking for trigger steps, then blocks

  BLOCK_LEVEL,       -- looking for trigger steps,
                     -- then blocks and/or fields

  FIELD_LEVEL );     -- looking for trigger steps,
                     -- then blocks and/or fields

  Level          : Level_type := APP_LEVEL;

  Step_Node   : Step_Node_type;
  Field_Node  : Field_Node_type;
  Block_Node  : Block_Node_type;
  App_Node    : App_Node_type;

  The_File       : Text_IO.File_type;
  The_Line       : Line_type;
  End_Line       : natural;
```

```
  The_Pattern    : Line_type;
  End_Pattern    : natural;
  The_Data       : Line_type;
  End_Data       : natural := 0;

  First_Char     : natural := 0;
  Last_Char      : natural := 0;
  Line_Number    : natural := 0;
  Column_Number  : natural := 0;

  Stop_Scroll : character := ' ';
pragma page;
-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--
-- ----------------------------------------------------------------------
  procedure Scan_Line(The_Pattern    : Line_type;
                      Pattern_Length : natural;
                      The_Line       : Line_type;
                      End_Line       : natural;
                      The_Data       : out Line_type;
                      End_Data       : out natural) is

    Hit, Column : natural := 0;

  begin

    The_Data := (others => ' ');

    Hit:=GREP.Location_Of
          (The_Pattern(1..Pattern_Length),The_Line(1..End_Line));

    if Hit > 0

      then

        for i in 1..End_Line loop

          if The_Line(i) /= ' '

            then
              Column := Column + 1;
              The_Data(Column) := The_Line(i);

            end if;

        end loop;
        End_Data := Column;

      else
        End_Data := 0;

      end if;
```

```
  exception

    when GREP.Pattern_Not_Found =>
       End_Data := 0;

  end Scan_Line;
-------------------------------------------------------------------------
pragma page;
-- +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--      Part_Of_App
-- -------------------------------------------------------------------
  function Part_Of_App(Current_Line       : in Line_type;
                       Current_line_End   : in natural)
                                            return Part_type is

    The_Data : Line_type;
    End_Data : natural;
    Part : Part_type := Null_Part;

    For_Step_Length     : natural := 7;
    For_Step_Number     : Line_type;

    For_Trigger_Length : natural := 6;
    For_Trigger_Name    : Line_type;

    For_Field_Length    : natural := 18;
    For_Field_Name      : Line_type;

    For_Block_Length    : natural := 18;
    For_Block_Name      : Line_type;

    Start_Here : natural;

pragma page;
  begin


    For_Step_Number(1..For_Step_Length) := "STEP = ";
    Scan_Line(For_Step_Number, For_Step_Length,
              Current_Line, Current_Line_End, The_Data, End_Data);
    if End_Data > 0
      then
        Part := STEP_NUMBER_PART;
        for i in reverse 1..Max_Name_Length loop
          if Step_Node.Name(i) = ' '
            then Start_Here := i+2;
            end if;
        end loop;
        Step_Node.Name(Start_Here..Max_Name_Length) :=
                The_Data(1..Max_Name_Length-Start_Here+1);
      end if;
```

```
    For_Trigger_Name(1..For_Trigger_Length) := "TYPE: ";
    Scan_Line(For_Trigger_Name, For_Trigger_Length,
              The_Line, End_Line, The_Data, End_Data);
    if End_Data > 0
      then
        Part := TRIGGER_NAME_PART;
        Step_Node.Name := The_Data(1..Max_Name_Length);
      end if;
pragma page;

    For_Field_Name(1..For_Field_Length) := "FIELD [0123456789]";
    Scan_Line(For_Field_Name, For_Field_Length,
              The_Line, End_Line, The_Data, End_Data);
    if End_Data > 0
      then
        Part := FIELD_PART;
        Field_Node.Name := The_Data(1..Max_Name_Length);
      end if;



    For_Block_Name(1..For_Block_Length) := "Block [0123456789]";
    Scan_Line(For_Block_Name,For_Block_Length,
              The_Line,End_Line, The_Data, End_Data);
    if End_Data > 0
      then
        Part := BLOCK_PART;
        Block_Node.Name := The_Data(1..Max_Name_Length);
      end if;


    return Part;

  end Part_Of_App;
-------------------------------------------------------------------------
pragma page;
-------------------------------------------------------------------------
begin  -- FP2SX

  App_Node := App_List.Head_Of(App);

  for i in 1..Max_File_Name_Length loop
    if App_Node.Name(i) /= ' '
      then
        First_Char := i;
        exit;
      end if;
  end loop;

  for i in reverse 1..Max_File_Name_Length loop
    if App_Node.Name(i) /= ' '
      then
```

```
            Last_Char := i;
            exit;
          end if;
        end loop;

        Text_IO.New_Line;
        Text_IO.Put("Opening " & App_Node.Name(First_Char..Last_Char));
        Text_IO.Put(", and reading");
        Text_IO.New_Line;
        Text_IO.Open
          (The_File, Text_IO.In_File,
           App_Node.Name(First_Char..Last_Char) );


        Thru_File:
        while not Text_IO.End_Of_File(The_File) loop

          Text_IO.Get_Line(The_File, The_Line, End_Line);
          Line_Number := Line_Number + 1;

          if Line_Number mod 100 = 0
            then
              Text_IO.Put("Line #" & natural'image(Line_Number));
              Text_IO.New_Line;
            end if;

      pragma page;
          case Part_Of_App(The_Line, End_Line) is

            when BLOCK_PART =>
              Block_Node.Steps  := Step_List.Null_list;
              Block_Node.Fields := Field_List.Null_List;
              Block_List.Construct(The_Item=>Block_Node ,
                                   And_The_List=>App_Node.Blocks );
              App_Node.UFP(TOTAL,ENTITIES) :=
                App_Node.UFP(TOTAL,ENTITIES) + 1;
              Level := BLOCK_LEVEL;

            when FIELD_PART =>
              Field_Node.Steps := Step_List.Null_List;
              Field_List.Construct(The_Item=>Field_Node ,
                                   And_The_List=>Block_Node.Fields );
              App_Node.UFP(TOTAL,OUTPUTS) :=
                App_Node.UFP(TOTAL,OUTPUTS) + 1;
              Level := FIELD_LEVEL;

            when STEP_NUMBER_PART =>
              case Level is

                when App_LEVEL  =>  -- add to App_STEPS
                  Step_List.Construct(The_Item=>Step_Node ,
                                      And_The_List=> App_Node.Steps);
```

```
            App_Node.UFP(TOTAL,INPUTS) :=
              App_Node.UFP(TOTAL,INPUTS) + 1;

          when BLOCK_LEVEL =>  -- add to BLOCK_STEPS
            Step_List.Construct(The_Item=>Step_Node ,
                                And_The_List=> Block_Node.Steps);
            App_Node.UFP(TOTAL,INPUTS) :=
              App_Node.UFP(TOTAL,INPUTS) + 1;

          when FIELD_LEVEL =>  -- add to FIELD_STEPS
            Step_List.Construct(The_Item=>Step_Node ,
                                And_The_List=> Field_Node.Steps);
            App_Node.UFP(TOTAL,INPUTS) :=
              App_Node.UFP(TOTAL,INPUTS) + 1;

        end case;

      when TRIGGER_NAME_PART    => null;

      when NULL_PART            => null;

    end case;
pragma page;
    if not Field_List.Is_Null(Block_Node.Fields)

      then
        Field_List.Set_Head(Of_The_List=>Block_Node.Fields,
                            To_The_Item=>Field_Node);
      end if;

    if not Block_List.Is_Null(App_Node.Blocks)

      then
        Block_List.Set_Head(Of_The_List=>App_Node.Blocks,
                            To_The_Item=>Block_Node);
      end if;

  end loop Thru_File;

    for j in SW_Function_type loop
      Natural_IO.Put(App_Node.UFP(TOTAL,j));
    end loop;

    --Stop_Scroll := Get_Option("Any key ... ");

  App_List.Set_Head (Of_The_List => App,
                     To_The_Item => App_Node);

  Text_IO.Close(The_File);
  Text_IO.New_Line;
  Text_IO.Put("Closing " & App_Node.Name(First_Char..Last_Char));
  Text_IO.New_Line;
```

```
exception

  when Text_IO.Name_Error =>

       Text_IO.New_Line;
       Text_IO.Put(App_Node.Name & "not found.");
       Text_IO.New_Line;

end FP2SX;
-------------------------------------------------------------------------


-- -----------------------------------------------------------------------
pragma page;

separate(FP2) procedure FP2V(Project : in out Project_List.List) is
  pragma Optimize(Time);

--

Project_Node : Project_Node_type;
App_Node     : App_Node_type;
Last_Char    : natural := 0;
SQL_Filename : File_Name_type := (others => ' ');


--

FP2V_Output    : Text_IO.File_type;
FP2V_SQL_File : Text_IO.File_type;

--

procedure Save_Steps(Steps : in Step_List.List) is
  Step_Node : Step_Node_type;
begin
  if not Step_List.Is_Null(Steps)
    then
      Save_Steps(Step_List.Tail_Of(Steps));
      Text_IO.New_Line;
      Step_Node := Step_List.Head_Of(The_List => Steps);
      Text_IO.Put(Step_Node.Name);
  end if;
end Save_Steps;

--
procedure Save_Fields(Fields : in Field_List.List) is
  Field_Node    : Field_Node_type;
begin
  if not Field_List.Is_Null(Fields)
    then
      Save_Fields(Field_List.Tail_Of(Fields));
      Text_IO.New_Line;
```

```
      Field_Node := Field_List.Head_Of(The_List => Fields);
      Text_IO.Put(Field_Node.Name);
      if not Step_List.Is_Null(Field_Node.Steps)
        then
          Save_Steps(Field_Node.Steps);
        end if;
      end if;
end Save_Fields;
pragma page;
--
procedure Save_Blocks(Blocks : in Block_List.List) is
  Block_Node    : Block_Node_type;
begin
  if not Block_List.Is_Null(Blocks)
    then
      App_Node.UFP(TOTAL,ENTITIES) :=
        App_Node.UFP(TOTAL,ENTITIES) + 1;
      Save_Blocks(Block_List.Tail_Of(Blocks));
      Text_IO.New_Line;
      Block_Node := Block_List.Head_Of(The_List => Blocks);
      Text_IO.Put(Block_Node.Name);
      if not Step_List.Is_Null(Block_Node.Steps)
        then Save_Steps(Block_Node.Steps);
        end if;
      if not Field_List.Is_Null(Block_Node.Fields)
        then Save_Fields(Block_Node.Fields);
        end if;
      end if;
end Save_Blocks;

--
procedure Save_Apps(Apps : in App_List.List) is
begin
  if not App_List.Is_Null(Apps)
    then
      App_Node.UFP(TOTAL,TRANSACTIONS) :=
        App_Node.UFP(TOTAL,TRANSACTIONS) + 1;
      Save_Apps(App_List.Tail_Of(Apps));
      Text_IO.New_Line;
      App_Node := App_List.Head_Of(The_List => Apps);
      Text_IO.Put(App_Node.Name);
      if not Step_List.Is_Null(App_Node.Steps)
        then Save_Steps(App_Node.Steps);
        end if;
      if not Block_List.Is_Null(App_Node.Blocks)
        then Save_Blocks(App_Node.Blocks);
        end if;
      end if;
end Save_Apps;

procedure SQL (Project_record : in Project_Node_type) is
--
```

```
begin
    Text_IO.New_Line;
    Text_IO.Put("INSERT INTO FP2_SER VALUES");
    Text_IO.New_Line;
    Text_IO.Put("('" & Project_record.SER_NUM & "',");
    Text_IO.Put("'"  & Project_record.Team_ID(1..12) & "',");
    Text_IO.New_Line;

    Text_IO.Put("'");
        FloatPt_IO.Put(Project_record.Code_Wks);
        Text_IO.Put("',");

    Text_IO.Put("'");
        FloatPt_IO.Put(Project_record.Test_Wks);
        Text_IO.Put("',");

    Text_IO.Put("'");
        FloatPt_IO.Put(Project_record.Code_Hrs);
        Text_IO.Put("',");

    Text_IO.Put("'");
        FloatPt_IO.Put(Project_record.Test_Hrs);
        Text_IO.Put("',");
    Text_IO.New_Line;

    Natural_IO.Put(Project_record.UFPs(ADDED,TRANSACTIONS)  );
      Text_IO.Put(",");
    Natural_IO.Put(Project_record.UFPs(CHANGED,TRANSACTIONS));
      Text_IO.Put(",");
    Natural_IO.Put(Project_record.UFPs(DELETED,TRANSACTIONS));
      Text_IO.Put(",");

    Natural_IO.Put(Project_record.UFPs(ADDED,ENTITIES));
      Text_IO.Put(",");
    Natural_IO.Put(Project_record.UFPs(CHANGED,ENTITIES));
      Text_IO.Put(",");
    Natural_IO.Put(Project_record.UFPs(DELETED,ENTITIES));
      Text_IO.Put(",");

    Natural_IO.Put(Project_record.UFPs(ADDED,OUTPUTS));
      Text_IO.Put(",");
    Natural_IO.Put(Project_record.UFPs(CHANGED,OUTPUTS));
      Text_IO.Put(",");
    Natural_IO.Put(Project_record.UFPs(DELETED,OUTPUTS));
      Text_IO.Put(",");

    Natural_IO.Put(Project_record.UFPs(ADDED,INPUTS));
      Text_IO.Put(",");
    Natural_IO.Put(Project_record.UFPs(CHANGED,INPUTS));
      Text_IO.Put(",");
    Natural_IO.Put(Project_record.UFPs(DELETED,INPUTS));
      Text_IO.Put(");");
```

```
    Text_IO.New_Line;

end SQL;
pragma page;
begin

  if Project_List.Is_Null(Project)

    then
      Text_IO.Put("No projects to save.");

    else

      Project_Node := Project_List.Head_Of(The_List => Project);

      for i in reverse 1..SER_Num_Length loop

        if Project_Node.SER_Num(i) /= ' '
          then
            Last_Char := i;
              exit;
          end if;

      end loop;

      Text_IO.Create(File => FP2V_Output,
                     Name => "fp2old." &
                     Project_Node.SER_Num(1..Last_Char) );
      Text_IO.Set_Output(File => FP2V_Output);

      for i in 1..72 loop
        Text_IO.Put("=");
      end loop;

     Text_IO.Put("SER #"  & Project_Node.SER_Num   & ", ");
     Text_IO.Put("Team: " & Project_Node.Team_ID);
     Text_IO.New_Line;

      for i in 1..72 loop
        Text_IO.Put("-");
      end loop;

      if not App_List.Is_Null(Project_Node.Apps)
        then
          Save_Apps(Project_Node.Apps);
        end if;

      SQL_Filename(1..3) := "ser";
      for i in 1 .. SER_Num_Length loop
        if Project_Node.SER_Num(i) = ' '
          then
            Last_Char := i;
```

```
            exit;
          end if;
        end loop;

        SQL_Filename(1..7+Last_Char) := "ser" &
          Project_Mode.SER_Num(1..Last_Char)  & ".sql";

        Text_IO.Create(File => FP2V_SQL_File,
                       Name => SQL_Filename);
        Text_IO.Set_Output(File => FP2V_SQL_File);

        SQL(Project_Mode);

        Text_IO.Set_Output(File => Text_IO.Standard_Output);

        Text_IO.New_Line;
        Text_IO.Put
        ("Now, login to SQLPLUS and START SER" & Project_Mode.SER_Num);

      end if;

end FP2V; -- in spite of salieri
-----------------------------------------------------------------------
```

Bibliography

1. Albrecht, Allan J. "Measuring Application Development Productivity," in *Proceedings Joint SHARE/GUIDE/IBM Application Development Symposium* 1979.

2. Albrecht, Allan J. and J. E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering 9*: 639–648 (November 1983).

3. Bailor, Maj. Paul and Maj. J. Howatt, "Class Lecture in CSCE595 Software Generation and Maintenance." School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, January 1992.

4. Behrens, Charles A., "Measuring the Productivity of Computer Systems Development Activities with Function Points," *IEEE Transactions on Software Engineering 9*: 648–649 (November 1983).

5. Blanken, Henk, "Implementing Version Support for Complex Objects," *Data and Knowledge Engineering 6*: 1–25 (January 1991).

6. Boehm, Barry W., *Software Engineering Economics*. Englewood Cliffs NJ: Prentice-Hall, 1981.

7. Booch, Grady, *Software Components with Ada: Structures, Tools, and Subsystems*. Menlo Park CA: Benjamin/Cummings, 1986.

8. Brown, Darlene, "Productivity Measurement Using Function Points," *Software Engineering* 13: 23–32 (July 1990).

9. D'Angelo, Dr. Anthony, "Class Lecture in AMGT602 Federal Financial Management." School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, August 1992.

10. DeMarco, Thomas and T. Lister, "State of the Art Paper," 1990.

11. Deming, W. Edwards, *Out of the Crisis*. Cambridge, MA: Massachusetts Institute of Technology, 1986.

12. Dreger, J. Brian, *Function Point Analysis*. Englewood Cliffs NJ: Prentice-Hall, 1989.

13. Esterling, Bob, "Software Manpower Costs: A Model," *Datamation 13*: 164–170 (March 1980).

14. Gould, J. P. and C. E. Ferguson, *Microeconomic Theory*. Homewood IL: Irwin, 1980.

15. Henry, S. and D. Kafura, "Software Structure Metrics Based on Information Flow," *IEEE Transactions on Software Engineering 7*: 510–518 (May 1981).

16. Henry, Sallie and J. Lewis, "Integrating Metrics into a Large-Scale Software Development Environment," *The Journal of Systems and Software 13*: 89–96 (October 1990).

17. Humphrey, Watts S. and N. D. Singpurwalla, "Predicting (Individual) Software Productivity," *IEEE Transactions on Software Engineering 17*: 196–207 (February 1991).

18. Jones, Capers, *Applied Software Measurement*. New York: McGraw-Hill, 1991.

19. Low, Graham C. and D. R. Jefery, "Function Points in the Estimation and Evaluation of the Software Process," *IEEE Transactions on Software Engineering 16*: 64–71 (January 1990).

20. McCabe, Thomas J., "A Complexity Measure," *IEEE Transactions on Software Engineering 2*: 308–320 (April 1976).

21. Neter, John, W. Wasserman and M., H. Kutner, *Applied Linear Regression Models*. Homewood IL: Irwin, 1989.

22. Oracle Corporation, Belmont CA, *Pro*Ada User's Guide* November 1986.

23. Pirsig, Robert M., *Zen and the Art of Motorcycle Maintenance*. New York NY: William Morrow and Co., 1974.

24. Putnam, Lawrence H., *Measures for Excellence*. Englewood Cliffs NJ: Prentice-Hall, 1992.

25. Rakos, John J., *Software Project Management for Small to Medium Sized Projects*. Englewood Cliffs NJ: Prentice-Hall, 1990.

26. Ratcliff, Bryan and A. L. Rollo, "Adapting Function Point Analysis to Jackson System Development," *Software Engineering Journal* (January 1990).

27. Reynolds, Daniel, "Class Lecture in MATH696 General Linear Model." School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1992.

28. SAS Institute, Cary NC, *SAS Procedures Guide* 1988.

29. SAS Institute, Cary NC, *SAS User's Guide : Statistics Version 6 ed* 1991.

30. Sayles, Jonathon S., *How to Use Oracle SQL*Plus*. Wellesley MA: QED Information Sciences, 1989.

31. Sommerville, Ian, *Software Engineering*. London, England: Addison-Wesley, 1989.

32. Symons, Charles R., "Function Point Analysis: Difficulties and Improvements," *IEEE Transactions on Software Engineering 14*: 2–11 (January 1988).

33. Symons, Charles R., *Software Sizing and Estimating Mk II FPA*. West Sussex, England: John Wiley and Sons Ltd, 1991.

34. Systems Support Division, "Thesis Topic," 1990. Air Force Institute of Technology (AU), Wright-Patterson AFB OH.

35. Systems Support Division, "Mission Statement," 1991. Air Force Institute of Technology (AU), Wright-Patterson AFB OH.

36. Tanenbaum, Andrew S., *Structured Computer Organization*. Englewood Cliffs NJ: Prentice-Hall, 1984.

37. vanGenuchten, Michiel, "Why is Software Late? An Empirical Study of Reasons For Delay in Software Development," *IEEE Transactions on Software Engineering 17*: 582–589 (June 1991).

38. Verner, June and G. Tate, "Estimating Size and Effort in Fourth-Generation Development," *IEEE Software 5*: 15–22 (July 1988).

Vita

Captain Steven D. Radnov was born on 20 December 1957 in Omaha, Nebraska. He graduated from William Jennings Bryan High School in Omaha, Nebraska in 1976. He enlisted in the U.S. Air Force in 1978 as an Accounting and Finance technician and was stationed at Lackland Air Force Base, Texas from January 1979 to April 1983. He was accepted into the Airman's Education and Commissioning Program and earned a bachelor's degree in Computer Science from the University of Nebraska at Omaha, and was commissioned at Officer Training School in August of 1985. He was stationed at Offutt AFB, Nebraska from September 1985 until April 1991. At Offutt AFB, he was a database analyst in the Intelligence Support Directorate and was Chief of the Imagery Exploitation Section. He was selected to pursue a master's degree in Software Systems Management at the Air Force Institute of Technology.

Permanent Address: 5818 Orchard Ave. Omaha, NE 68117

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1992	Master's Thesis

4. TITLE AND SUBTITLE
SOFTWARE SUPPORT MEASUREMENT AND ESTIMATING FOR ORACLE DATABASE APPLICATIONS USING MARK II FUNCTION POINTS

5. FUNDING NUMBERS

6. AUTHOR(S)
Steven D. Radnov, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GSS/LSY/92D-4

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Captain Auzenne
AFIT/SCV
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This study investigated the results of measuring software support of Oracle database applications and estimating the effort and schedule required to provide support. Software measurement was accomplished with a variant of the function points metric, called Mark II function points, which is comprised of three weighted parameters, inputs, entities, and outputs. A technique for mapping Mark II function points to Oracle DBMS components was developed, and the size of the software support for each project, per team, was measured by tabulating and weighting the number of inputs, entities, and outputs that are added, changed, and/or deleted. Software support effort was measured in work-hours and schedule in calendar-weeks for given levels of function points. A data collection program was written to assist with tabulating the measurements and also provided an option for sizing the support by analogy. Observations were collected for 12 projects ranging up to 50 function points. The relationship between software support measurement in Mark II function points and the resulting effort or schedule was extensively analyzed for one and two person teams. A relationship determined by regression analysis was shown to be statistically significant for both effort and schedule.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software, Software Development, Software Measurement, DBMS, Software Maintenance, Function Points, Software Estimation, Databases, Database Development	150
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT Control Number AFIT/GSS/LSY/92D-4

AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to determine the potential for current and future applications of AFIT thesis research. Please return completed questionnaires to: AFIT/LSC, Wright-Patterson AFB OH 45433-9905.

1. Did this research contribute to a current research project?

a. Yes b. No

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not researched it?

a. Yes b. No

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency received by virtue of AFIT performing the research. Please estimate what this research would have cost in terms of manpower and/or dollars if it had been accomplished under contract or if it had been done in-house.

Man Years _____ $ _____

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3, above) what is your estimate of its significance?

a. Highly Significant b. Significant c. Slightly Significant d. Of No Significance

5. Comments

_____ _____

Name and Grade Organization

_____ _____

Position or Title Address